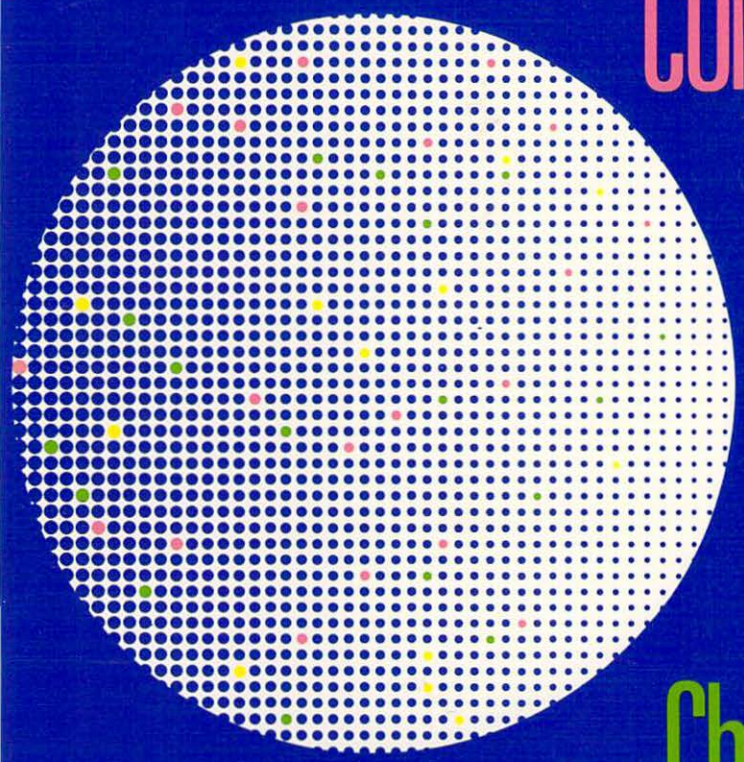
Using MS-DOS® KERMIT



CONNECTING YOUR PC TO THE ELECTRONIC WORLD

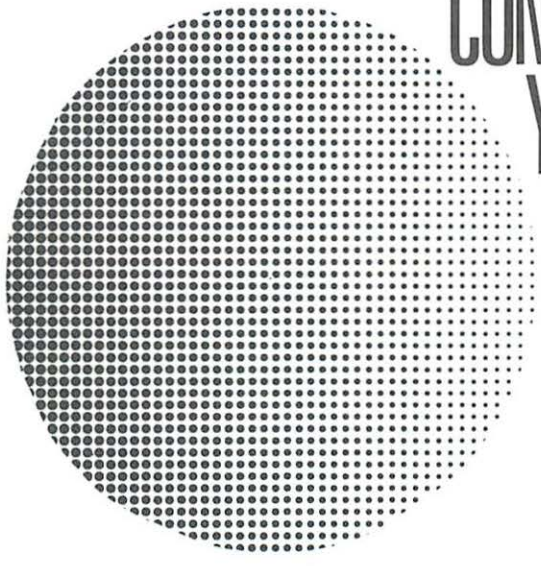
Christine M. Gianone

Terminal Emulation
File Transfer

Inside!
5¼-inch
Kermit
Diskette

Using MS-DOS KERMIT

Using MS-DOS KERMIT



CONNECTING YOUR PC TO THE ELECTRONIC WORLD

Christine M. Gianone

digital
Digital Press

9 8 7 6 5 4 3 2 1

Order number EY-C204E-DP

Printed in the United States of America.

Trademarks and trademarked products mentioned in this book are listed on pages 235 and 236.

Design: Sandra Calef
Copyediting: Robert Fiske
Illustration: Carol Keller
Index: Gordon Brumm
Production: Editorial Inc.
Printing and binding: Hamilton Printing Company

Library of Congress Cataloging-in-Publication Data

Gianone, Christine M.
Using MS-DOS Kermit : connecting your PC to the electronic world / Christine M. Gianone.
p. cm.
Includes bibliographical references.
ISBN 1-55558-048-3
1. Communications software. 2. MS-DOS (Computer operating system)
I. Title.
TK5105.9.G5 1989 89-49550
005.7'1369—dc20 CIP

This book is lovingly dedicated to my parents, Sal and Phyllis,
who nurtured me with encouragement,
support and love throughout my life.

Contents

Illustrations

Tables

Foreword

It is with tangible joy that I introduce Christine Gianone's book, *Using MS-DOS Kermit*. This is the first book dedicated to a particular Kermit communications program, and it truly brings Kermit into the realm of the serious. Until now, users of Kermit programs have had only a thick sheaf of computer output to guide them through the intricacies of installation, communication setup, terminal emulation, file transfer, and script programming. Now MS-DOS Kermit, the most popular of all Kermit programs, has the book it deserves.

Because of its unglamorous user interface, MS-DOS Kermit may appear to the uninitiated as a no-frills product. Believe me, the frills are there, but beneath the surface where you really need them. They are found in its precision-engineered character and graphics terminal emulation, its support for every model of PC and PS/2 as well as for many non-IBM-compatible PCs, its high-speed and efficient operation, its powerful macros and scripts, and in one of the most advanced and solid implementations of the Kermit file transfer protocol to be found anywhere. Compare these aspects of MS-DOS Kermit with any commercial PC communications software package and you'll be pleasantly surprised, especially when you consider the price!

Credit for MS-DOS Kermit goes to Professor Joe R. Doupnik of Utah State University, the principal author of the program. Since 1986, when he took over responsibility for MS-DOS Kermit, Joe, like all Kermit volunteer programmers, has generously donated his labors to the public through Columbia's Kermit Development and Distribution Office, which is directed by Christine, who in turn releases the Kermit software to the people of the world. Joe's work is of the highest quality, with a sensitivity to users' needs that is seldom seen among nonprofit software developers.

Chris has been responsible for the worldwide Kermit development and distribution effort for more than four years. In that time she has transformed a public-spirited but chaotic enterprise into an efficient nonprofit "business" as responsive to its customers as any commercial software house (more than most). Chris's dedication, her persuasive charm and wit, her unerring good sense, and her uncanny feeling for what users *need* have kept unruly Kermit developers (like me) firmly on track, and Kermit's countless users satisfied and loyal. And through magazine articles and press releases, seminars and lectures in far-flung areas of the world, Chris has effectively promoted the Kermit protocol into a worldwide de facto standard for file transfer.

Chris's book is for the average PC user—a person who doesn't care how Kermit works, how clever it is, or what goes on behind the scenes; a person whose life does not revolve around data communications, but who simply needs results. After all, Kermit is just a tool that you use to accomplish your actual goals. With this book, learning how to use the tool is a true delight.

Chris's presentation is based on years of helping people install and use Kermit, and on her experience designing Kermit courses and teaching beginners, experts, and everyone in between, and on her past authorship of many of the major Kermit manuals. She is expert in the material, and has an amazing talent for paring difficult concepts down to their bare essentials and conveying them simply and directly—especially to people who don't have that peculiar technical orientation that most PC communication packages seem to require.

If you're an experienced PC user, you can skip right ahead to Chapter 6 to learn about Kermit itself. But if you are new to PCs and data communications, computer-shy, bewildered, lost, read the early chapters in which Chris takes you through the mechanics of program installation and checkout, modems and cables, and the basics of MS-DOS—all with a minimum of pain and effort. After these preliminaries, *Using MS-DOS Kermit* will lead you gently and safely through the unpredictable and often hostile world of data communications without drowning you in jargon or confusing you with unnecessary detail.

Chris's instructions are clear, concise, and step-by-step, and they are presented with a humor and sympathy rarely encountered in computer books. Useful examples appear on nearly every page, so you can build your confidence and increase your skill as you progress through the chapters. Once the material is mastered, the book remains a thoughtfully organized and valuable reference. The command summary is thorough and replete with examples. The meticulously compiled tables alone are worth the price of the book. And the Index gives you quick access to any desired topic.

Chris's book has a unique international perspective. Her travels as "proprietor" of the non-proprietary Kermit protocol have taken her not only throughout the United States, but also to Europe, Japan, and most recently the Soviet Union. In response to the needs of the

non-English-speaking world, Chris has designed a groundbreaking extension to the Kermit protocol that allows for entry, display, and transfer of text in many languages; a complex topic that she describes with remarkable clarity in Chapter 13.

Another unique aspect of this book, and of MS-DOS Kermit itself, is its concern for the disabled computer user. MS-DOS Kermit, unlike most other "mass-market" PC applications, includes features that allow it to be used effectively by people who are partially sighted, blind, deaf, or physically impaired. These are presented by Chris simply and without fanfare in Chapter 15.

It has been a constant pleasure to work with two such accomplished and inspiring people as Christine and Joe. Without Joe's efforts, MS-DOS Kermit would be a dusty old hasbeen, and without Chris, the Kermit protocol itself would have faded into obscurity years ago. On behalf of the millions of Kermit users all over the world, my deepest thanks to them both.

Frank da Cruz
New York City, 1990

Preface

If you picked up this book and began reading it because you wondered why a muppet character's name was in the computer section of the bookstore, you'll soon discover that this book is not about muppets: *Kermit* is the name of a communications software program. In fact, it's the name of a very large family of programs that make it possible for just about any two computers anywhere to communicate with each other at the lowest possible cost. If this interests you, read on.

Using MS-DOS Kermit describes the MS-DOS Kermit software for the IBM PC, PS/2, and compatibles. With MS-DOS Kermit, a cable, and possibly a modem, you can introduce your isolated PC to the rest of world: to information and electronic mail services like CompuServe, Dow Jones News/Retrieval, or MCI Mail, to databases like BRS or DIALOG; to public data networks like Telenet and TYMNET; and to the worldwide networks described in John Quarterman's book, *The Matrix*.[1] But most of all, you can introduce your PC to other computers: the PC on your desk to the corporate mainframe, your PC at home to the computers at work, your traveling laptop back to company headquarters, the PC in your dormitory room to the university computer center, the PC in your lab to a number-crunching supercomputer. Kermit can be your passport to the world of electronic information.

---

[1]Quarterman, John S., *The Matrix*, Digital Press (1990).

MS-DOS Kermit is only one of hundreds of Kermit programs written by public-spirited volunteers for virtually every kind of computer in existence. Each Kermit program follows the Kermit file transfer protocol, a set of rules for reliably transferring information between two computers, and many of these programs, including MS-DOS Kermit, also let you interact directly with other computers or services using your keyboard and screen for terminal emulation.

This book includes a 5.25-inch MS-DOS Kermit diskette. If you need a 3.5-inch diskette, you may order it for a nominal fee, using the card at the back of this book. For further information about Kermit programs and how to get them, or to be placed on the mailing list for the free newsletter, *Kermit News*, write to:

Kermit Distribution
Columbia University
Center for Computing Activities
612 West 115th Street
New York, NY 10025
USA

Kermit programs are in a category by themselves. Unlike commercial software packages or "shareware," Kermit programs can be freely reproduced and shared as long as this is not done for profit. No licensing or registration is required. However, Kermit programs are not in the public domain. In general, each Kermit program includes a copyright notice to protect its special status. A large organization can potentially save itself millions of dollars by standardizing on Kermit rather than on commercial software packages, and many organizations have done just that. And, of course, private individuals save money too.

The Kermit protocol was developed in 1981 at Columbia University by Frank da Cruz and his staff at the Center for Computing Activities to fill the need for inexpensive, reliable file transfer between Columbia's diverse PCs, minicomputers, and mainframes. Nearly a decade later, Frank is still very much involved in the "Kermit project" and is the principal author of another popular Kermit program, C-Kermit.

Kermit software is contributed by programmers in many countries and has become the international de facto standard for file transfer. Today there are Kermit programs for about 400 different computers and operating systems, and the number is constantly growing. With very few exceptions, Kermit will let you transfer files between your PC and virtually any other kind of computer.

The Kermit program for the IBM PC was created by the systems programming group at Columbia University in 1982, managed by Frank. The groundwork was laid by Bill Catchings, based on his original CP/M Kermit program. Daphne Tzoar wrote the first

production version and, together with Jeff Damens, produced subsequent releases through 2.28 in 1985. In 1986, MS-DOS Kermit development was taken over by Professor Joe R. Doupnik of the Center for Atmospheric and Space Sciences and the Department of Electrical Engineering at Utah State University in Logan, Utah.

The current MS-DOS Kermit program, Version 3.0, is one of the most advanced examples of the Kermit software. It includes error-free and efficient file transfer, advanced terminal emulation, graphics, a script programming language, operation over local area networks, support for text in many languages, special features for the disabled, and much more. Over the years, thanks to Joe's tireless efforts and consummate skill, and to other contributors from around the world, and to the many prerelease testers, MS-DOS Kermit has become one of the most powerful, flexible, and easy-to-use of all PC communication packages.

Using MS-DOS Kermit applies to MS-DOS Kermit Version 3.0. Many of the features described here will not be found in earlier releases. If you are running an older release, you should be able to upgrade quite easily. Get a copy from a friend, download it from a computer bulletin board or network, or order a diskette from Columbia University or your local PC user group.

This book is designed to teach you step by step how to make effective use of MS-DOS Kermit on the IBM PC, PS/2, and compatible personal computers. Most of what you read here will also apply to the non-IBM-compatible MS-DOS PCs that have Kermit programs available. Advanced and highly technical material is omitted. For an overview of data communications and a complete reference on the Kermit protocol itself, see the book *Kermit, A File Transfer Protocol* by Frank da Cruz, Digital Press, 1987. For technical details of MS-DOS Kermit, see *The Making of MS-DOS Kermit* by Joe R. Doupnik, Digital Press, 1990.

Welcome to the world of Kermit.

Acknowledgments

Special thanks to Frank da Cruz of Columbia University who opened the door for me to the world of Kermit. It is an honor to work with such a distinguished authority in the computer field. I am grateful for his recognition of my talents and the opportunities he has given me. Frank's encouragement, expertise, experience, and friendship were invaluable in the production of this book.

Special thanks also to Professor Joe R. Doupnik of Utah State University, whose skill and dedication resulted in a sophisticated software program and the subject of this book. Joe, with help from many others, has made MS-DOS Kermit a prime competitor in the PC communications software market.

Thanks to the the original MS-DOS Kermit programming staff, Daphne Tzoar, Jeff Damens, and Bill Catchings, formerly of Columbia University. And to the many others who have contributed to the program through the years: Bob Babcock (Harvard/Smithsonian Center for Astrophysics), Ron Blanford (Seattle, WA), Jack Bryans (California State University at Long Beach), Fritz Bütikofer (University of Bern, Switzerland), Edgar Butt (University of Maryland), Dick Carlton (The Open University, UK), Baruch Cochavy (IIT, Technion, Israel), Glenn Everhart (RCA, Cherry Hill, NJ), Hirofumi Fujii (Japan National Laboratory for High Energy Physics, Tokyo), Ian Gibbons (University of Hawaii), Robert Goeke (MIT Center for Space Research), James Harvey (Indiana/Purdue University), Brian Holley (University of Cambridge, England), Terry Kennedy (Saint Peters College, NJ), Dave King (Carnegie-Mellon University), David Knoell (Basic American Food Company), Kimmo Laaksonen (Helsinki University of Technology, Finland), Mikko Laanti (University of Oulu, Finland), Christopher Lent (Cooper Union), Henrik Levkowetz (Philips Kista AG, Stockholm) Ted Medin (NOSC), Geoff Mulligan (USAF Academy), Jim Noble (Planning Research Corporation), Dan Norstedt (Stacken Computer Club, Sweden), John Nyenhuis (Purdue University), Yutaka Ogawa (Nippon Telephone and Telegraph Research Lab, Tokyo), Brian Peterson (Brigham Young University), Renne Rehmann (Switzerland), Fred Richter (Intel Corporation, Hauppauge, NY), Joseph Rock (USAF Academy), Akihiro Shirahasi (Japan National Laboratory for High-Energy Physics, Tokyo), Gregg Small (University of California at Berkeley), Dan Smith and Joe Smith (Colorado School of Mines), Andreas Stumpf (University of Stuttgart, West Germany), James Sturdevant (A.C. Nielson Company), Glenn Trewitt (Stanford University), John Voigt (Tulane University), Joe White (Relational Technology Inc.), Joellen Windsor (University of Arizona), and Mark Zinzow (University of Illinois). And thanks to Bill Hall of Excelan Inc. for contributing the Heath character graphics files that are included on the distribution diskette.

Grateful acknowledgments to the specialists who have contributed to the proposal for extending the Kermit protocol to accommodate international character sets—a topic that has developed into a chapter in this book: John Chandler (Harvard/Smithsonian Center for Astrophysics, USA), Alan Curtis (University of London, UK), Frank da Cruz (Columbia University, USA), Joe Doupnik (Utah State University, USA), Hirofumi Fujii (Japan National Laboratory for High Energy Physics, Tokyo), John Klensin (Massachusetts Institute of Technology, USA), Ken-ichiro Murakami (Nippon Telephone and Telegraph Research Labs, Tokyo), Vladimir Novikov (VNIIPAS, Moscow, USSR), Jacob Palme (Stockholm University, Sweden), André Pirard (University of Liege, Belgium), Paul Placeway (Ohio State University, USA), Gisbert W. Selke (Wissenschaftliches Institut der Ortskrankenkassen, Bonn, West Germany), Fridrik Skulason (University of Iceland, Reykjavik), Johan van Wingen (Leiden, Netherlands), Konstantin Vinogradov (ICSTI, Moscow, USSR), and Amanda Walker (InterCon Systems Corp., USA). And to the programmers who are adding this protocol extension to new Kermit releases: John Chandler, Frank da Cruz, Joe Doupnik, Paul Placeway, and the "Kermit gang" at ICSTI.

Thanks to Apple Computer's Higher Education Academic Development Donations program for approving a grant to help fund this effort and to Columbia University's Apple account executive, Ron Bajakian, for his assistance.

Thanks also to Dr. Robert Arnzen, Jude T. DaShiell, Robert Jaquiss, Bob Puyear, Kenneth Reid, and others for providing information and suggestions about the use of Kermit by people with disabilities. Their comments have helped to make MS-DOS Kermit compatible with PCs and devices used by people with visual, hearing, or physical impairments.

Thanks to the people who have invited me as Kermit spokeswoman to their conferences and to their countries. These visits broadened my understanding of their data communication configurations as well as their cultures: Kohichi Nishimoto (Japan DECUS) and Ken-ichiro Murakami and Ikuo Takeuchi (Nippon Telephone and Telegraph Research Labs, Tokyo, Japan); David Gürlet (Switzerland DECUS); Juri Gornostaev, A. Butrimenko, Konstantin Vinogradov, Andrej Yuzhakov, Marina Tumanova, and Mikhail Morozov (ICSTI, Moscow, USSR); Chris Stephenson (Software Publishers Association); Bob McQueen (DECUS, Nashville, Tennessee) and guest speakers Brian Nelson (University of Toledo, Ohio), Joe Doupnik, and Frank da Cruz.

And to those not yet mentioned who were equally hospitable while I was a guest in their countries: Jean Dutertre and Jean-François Vibert (Club Kermit, Paris, France); Gisbert W. Selke (Bonn, West Germany); Marina and Vladimir Morozov (Moscow, USSR); Slava and Serge (Leningrad, USSR). Thanks also to my friend and travel agent Julie Balzer (Liberty Travel, New York) for handling the arrangements for these trips.

Thanks to the current staff of Kermit production for their dedication to the distribution of Kermit to users around the world: Maxwell Evarts, Kenneth Suh, and Takahiro Sajima. And to the generations of "Kermites" who came before: Robert Tschudi, Peter Howard, Lucy Lee, Moshe Mizrahi, Tara Garrett, David Suarez, Sarah Lewis, Adibah Abduljalil, Rajan Vohra, Paul Pincus, David Suh, David Ho, Phil Hsu, Chun-Lun Li, Guillermo Aries, Salman Mustafa, and Kyriacos Panayioton.

And thanks to the people who have established and maintain other Kermit distribution centers throughout the world: Dr. Hans-Magnus Aus (Institute for Virology and Immunobiology, Universität Würzburg, West Germany), Marc de Lyon and Frans-Jozef Springers (EARN Info Service, Netherlands), Jean Dutertre (Club Kermit, Paris, France), Juri Gornostaev and A. Butrimenko (ICSTI, Moscow, USSR), David Gürlet (DECUS, Bern, Switzerland), Ken-ichiro Murakami (Nippon Telephone and Telegraph Research Labs, Tokyo), Alan Phillips and Steve Jenkins (Lancaster University, UK), and Masamichi Ute (Science University of Tokyo, Japan).

Acknowledgments to Linda Ferreira, Gary Hanks, and Nina Frantzen of Columbia University's Division of Special Programs for inviting me to teach courses about Kermit and for preparing the class schedules and materials. The best way to thoroughly learn about a subject is to teach it to others.

Appreciation to my colleagues who have helped ease my professional transition from education to computing. Those who introduced me to the Center for Computing Activities at Columbia University: Bruce Gilchrist, Neil Sachnoff, Patricia Peters, Patrick Thompson, Sharon Moore, Terry Thompson, Anthony Bonfiglio, Fran Ovios, Bob Juckiewicz, Gary Platizky, Anna Harris, Peter Bujara, Sheila Greenbush, Robert Story, Hank Butler, Jim Cassidy, Ruth Rubinstein, Bob Bingham, Anne Simonsen, and Bill Koerber. Those who offered their consulting skills: Frank da Cruz, Maurice Matiz, Bruce Tetelman, David Millman, Don Lanini, Travis Winfrey, Ken Rossman, Vaçe Kundakçi, Jeff Damens, Mark Kennedy, Bill Chen, George Giraldi, Joel Rosenblatt, Howie Kaye, Fuat Baran, Melissa Metz, Robert Cartolano, Eric Weaver, Mark Lerner, Janet Asteroff, Jessica Gordon, Bob Miller, David Spital, and Janet Nelson.

Those who made my management training a rewarding and interesting experience: Leslie Wilkins, who has remained a dear friend, Reina Joa, Judith Weisenfeld, Carlos Jove, Becky Kaufman, Michelle Tzoar, and Sarah Bacon. And those who gave me my first introduction to data communications, even though I didn't understand the implications at the time: Bill Wojcik, Tom Hillery, Gary Williams, Peter Green, Lenny Wright, Bob Galanos, and Lon Devitt.

Thanks to Digital Press and those individuals who were part of the publication of this manuscript. Everyone involved worked skillfully under an extremely tight schedule: Mike Meehan, John Osborn, Chase Duffy, Beth French, Peg Tillery; the production editor, Ann Knight, and Laura Fillmore of Editorial Inc.; the thorough copy editor, Robert Fiske, and proofreaders, Barbara Jatkola and Stephani Colby; the talented book designer, Sandra Calef; the illustrator, Carol Keller; and the compositor, Frank da Cruz, a Scribe authority at Columbia's Computer Center.

Thanks to the people who reviewed this manuscript for their speed, enthusiasm, and thoughtful comments: Joe Doupnik and Frank da Cruz, two highly regarded Kermit experts, whose insights and cooperation were critical to achieving a balance between the software program and the book; Louis Santelli, for conveying the perspective of a naive user of data communications software and for his love and devotion; Sari Wilner, my dearest friend, who has read many computer software manuals and has reviewed my work before, tracing back to undergraduate college assignments; Salvatore Gianone, my brother, who has inspired me since childhood. Although Sal is an experienced computer user, he was forced to learn far more about hardware than expected when unknowingly struggling with two pieces of inoperable equipment: a bad serial port and a broken

modem. Unfortunately, this scenario is not as uncommon as you might think, and his experiences were a great help in writing two chapters in this book. Many thanks to Sal, Sari, and Louie for supporting my ventures over the years and for sharing my excitement in this project from the onset. Thanks also to Karen Rufa for her comments and suggestions.

Thanks to my family and friends who encouraged my writing endeavor and understood my time limitations while preparing this book. Special thanks to my parents for instilling in me the notion that nothing is impossible and to Roberta Gianone for her support and advice. Thanks too to Mary Rufa, Marilyn Wilner, and David Bronstein for their expressed confidence in my abilities.

And finally, my apologies to anyone I failed to mention in this long and international roster of people who have contributed to the development of the MS-DOS Kermit program and to the publication of this book. Thank you all.

Christine M. Gianone
New York City, 1990
cmg@columbia.edu
KERMIT@CUVMA.BITNET

Chapter 1

Introduction

Many of us are conditioned to read computer manuals only after a lot of trial and error. We are annoyed that the PC software, with all its menus and choices, wasn't self-explanatory enough to guide us. But sometimes there is simply no substitute for paper and ink. This book is here to ease the burden.

Communication software is a little more complicated than other PC-based software because *two* computers are involved instead of one. Not only are two computers involved, but these two computers must be successfully connected to each other before you can even *begin* to use a communication program. A Kermit program is even *more* complicated because it allows the other computer to be practically any make or model. But these complications allow for flexibility.

What Will Kermit Do for You?

PCs have become commonplace in both the office and the home. When PCs are able to communicate—to interact and share data—with each other, with the central mainframe computer, or with a dialup service, we can use them to provide a wider range of services.

Kermit has been used to explore outer space, to feed the hungry, and to heal the sick. It gathers data from scientific and medical devices. It transmits those amazing sports statistics to the television announcer at the game. It relays the computer commands that control giant steel furnaces and factory-floor milling machinery. It connects the PCs and com-

puters of most U.S. government agencies and most of the world's universities. It probably even runs in the cash register of your local fast-food restaurant.

Why is Kermit so popular? It's good and it's cheap. With Kermit, you'll be able to interact with nearby or distant time-sharing computers, large or small. You can exchange information safely and conveniently with almost any other kind of computer. You can access dialup services for news, stock quotations, electronic mail, banking, shopping, and file sharing. You can trade data with friends or neighbors, even if they have totally different kinds of computers. You can work from home. Kermit can connect you to "the network." Use your imagination. With today's worldwide telephone system, and our rapidly improving telecommunication and computing technology, there is no limit.

Capabilities of MS-DOS Kermit

MS-DOS Kermit is one of the most advanced and powerful of all the Kermit programs. Here is a list of some of its features for quick reference. Don't worry if you are unfamiliar with some of the terminology. You will have a better understanding of these capabilities as you read along.

- Error-correcting, efficient file transfer using the Kermit protocol. Text or binary files may be transferred singly or in groups.
- File transfer capabilities include long packets, sliding windows, international text character set conversion, checksum and CRC error correction, eighth-bit prefixing, attribute packets, server mode, data compression, file management functions, and more. MS-DOS Kermit also has the ability to transmit and receive files without error checking.
- Emulation of the DEC VT52, VT102, and VT320 terminals; of the Heath/Zenith-19 terminal; and of Tektronix graphics terminals. In VT320 mode, Kermit supports a wide variety of national character sets.
- Operation at speeds up to 57,600 bits per second (bps) or higher using a standard IBM communication port.
- Operation over a variety of local area networks, including Netbios, Ungermann-Bass Net/One, DECnet/DOS, and Intel OpenNET, and with Novell asynchronous communication servers.
- Compatibility with Microsoft Windows and similar operating environments.
- A wide variety of communication settings that can be matched to virtually any other computer.

- A simple, consistent, and intuitive command language with built-in help.
- Command and initialization files.
- Screen rollback, screen capture, and printer control.
- Key redefinition and keystroke macros.
- Command macros and a powerful script programming language.
- Logging, security, and debugging features.
- Special features for people with disabilities.

Kermit in a Nutshell

In Figure 1-1, we see a PC and a minicomputer. Each computer has something the other computer wants. The PC's disk is full of spreadsheets, databases, reports, messages, mailing lists, stories, and recipes that would be useful to the people who use the minicomputer. And the minicomputer has services the PC user needs: electronic mail, large-capacity disk storage, high-speed printers, networks, applications, and its own databases.

How does the PC user send information to the minicomputer, and how does the user access the minicomputer's applications? Files cannot be shared using disks or tapes since the minicomputer does not have a diskette drive, and the PC does not have a tape drive. The answer is *data communication*—sending data between the two computers through a wire. This is Kermit's job.

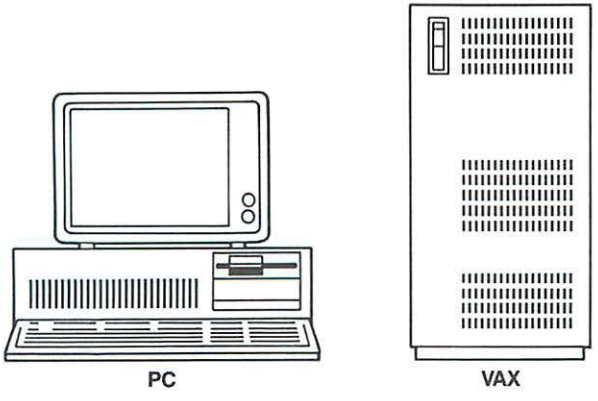


Figure 1-1 Two Computers in Search of a Connection

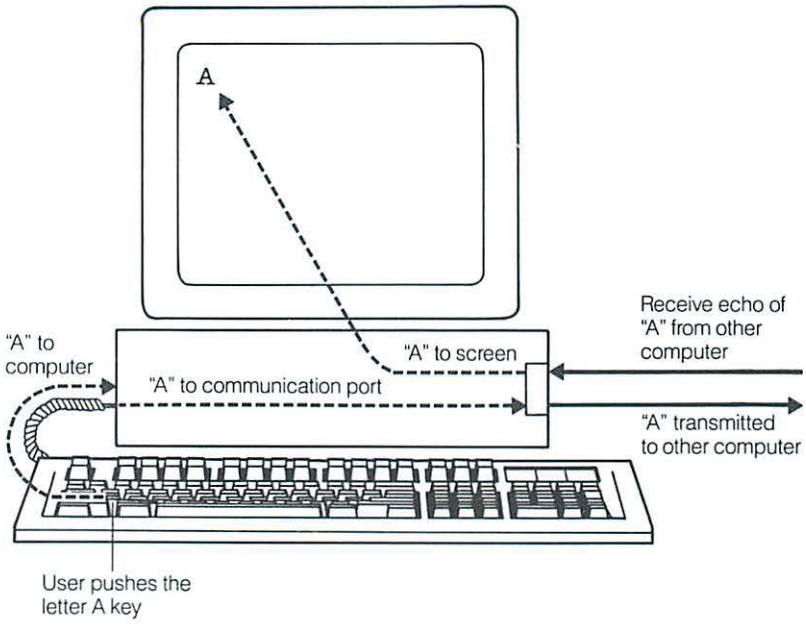


Figure 1-2 Terminal Emulation

MS-DOS Kermit does two things for you: *terminal emulation* and *file transfer*. Terminal emulation makes your PC act like a terminal so that you can carry on a dialog with another computer. At any time during this dialog, you can also send a file from your PC's disk to the other computer, or you can have the other computer send a file to your PC's disk.

Terminal emulation means that the characters (letters, numbers, and symbols) that you type on your PC's keyboard are sent through the PC's communication port to the other computer, and the characters that the other computer sends back are displayed on your screen, as shown in Figure 1-2. You have the illusion that the other computer is right in front of you.

This process, called *connect mode,* allows you to use applications on the other computer, such as electronic mail, text editors, databases, programming languages, and bulletin boards, just as if you had a real terminal. For most of the 1970s, almost all computing was done in this way. Many terminals were connected to a minicomputer or mainframe, and terminal users all shared the power and resources of the central computer.

Today, terminals are an endangered species. Most computer users now have personal computers (PCs) on their desks. A PC is a general-purpose computer, capable of perform-

ing most of the central computer's tasks and many new ones. PC users can create and print documents, balance budgets, manage databases, produce sophisticated reports, write programs in many languages, and so on. But there comes a time when a PC user needs to share his or her creations with friends or co-workers, or needs access to information that is on another computer. That's where data communications and Kermit come in.

A Failure to Communicate

In data communications, Murphy's Law, "Everything that can go wrong will go wrong," is an understatement. As the computer graffiti says, "Murphy was an optimist."

The two computers, your PC (the *local* computer) and the computer with which you are communicating (the *remote* computer) must be physically connected to each other. The connection between your PC and the other computer can be as simple as a direct cable or as complicated as a long-distance telephone call that is routed through modems, switching stations, microwave radio transmitters, undersea cables, and earth satellites. These connections are always susceptible to interference.

On a voice telephone call, static and other types of interference are not serious problems: You can usually make out what the other person is saying despite the noise. Computers, however, are not as smart as humans. They represent and transmit data using an extremely simple and vulnerable code consisting only of zeros and ones. Static can easily change a one to a zero or vice versa, and this will change your data. If you are sending next year's budget to the boss, you don't want the numbers corrupted by static (unless the error accidentally doubled your salary, but there are no guarantees in data communications).

The momentary silences that occur at critical parts of a telephone conversation, which blank out words or phrases, can occur in data communications too. Computers cannot necessarily tell that this has happened, and pieces of data could be missing in undetectable ways. Imagine, for example, if the word *don't* were deleted from the message "Now don't launch the missiles!"

And a phone call can go completely and undetectably dead. This can happen, for example, if the dog pulls your phone cord out of the wall during your conversation. Have you ever been talking to someone enthusiastically, on and on, when this happened? Fifteen minutes later, you pause to ask "Are you still there?" and nothing. If you call back and someone answers, maybe the lost connection was really an accident. The same thing can happen to a data connection, but usually without the dog.

Differences between the two people or computers that are communicating bring about other considerations besides the physical connection. Did you ever have someone dictating an address to you over the phone who talked faster than you could write it down? It is

perfectly natural for you to say "Slow down, please," and any normal, courteous person would. But it is not so simple with computers. If a huge and powerful mainframe is sending data to a small and frail PC's diskette, the PC might not be able to record the data as fast as it arrives, and it has no built-in, natural way of saying "Slow down."

And like people, not all computers speak the same language. Different computers can use different codes for storing and transmitting data. If you get a phone call from a person who doesn't speak your language, you're probably smart enough to realize it, even if it takes a few moments to sink in. A computer that receives data in an unknown code will most likely accept it but then interpret it incorrectly. The results could be disastrous.

File Transfer: Playing by the Rules

Because so much can go wrong with a phone call, and with the people at each end, a simple but effective set of rules and procedures has evolved over the years known as telephone etiquette, and it is a good model for data communications.

First, I initiate the connection by dialing your phone number. Your phone rings, you pick it up to complete the connection, and then you say "Hello?" I say hello back and identify myself. You either agree or refuse to talk to me. If you agree, you and I take turns talking. If you're going too fast, I ask you to slow down. If there is interference and I can't understand you, I ask you to repeat yourself. When we are finished talking, we say good-bye and hang up, which breaks the connection. If either of us hangs up before saying good-bye, it is a breach of etiquette.

When a set of rules and procedures grows sufficiently complicated and formal, it becomes a *protocol*. We are familiar with this term from diplomacy, a sphere in which people who might not share the same native language or customs must nevertheless communicate with clarity and precision.

Data communications protocols are even greater sticklers for detail than diplomats. They are the means by which computers of different sizes, shapes, speeds, operating systems, and codes and from different manufacturers and countries can meaningfully communicate. This is done by exchanging strictly formatted messages according to well-defined and agreed-on rules.

But computers do not do this on their own. People must formulate the rules and write computer programs like Kermit to carry them out. Kermit's rules for file transfer, illustrated in Figure 1-3, are similar to telephone etiquette:

1. Make the initial connection, say hello, and make sure the other computer is ready to transfer a file.
2. Tell the other computer the name of the file it will be getting.

3. Break the file up into smaller pieces called *packets*, and send one packet at a time.
4. The file receiver inspects each arriving packet for damages and says "yes" to accept it or "no" to reject it.
5. When a packet is accepted, the next one is sent. If a packet is rejected, it is sent again.
6. If the same packet is rejected too many times, or if an acknowledgment never arrives, the file transfer fails.
7. When the file has been completely transferred, the receiver is told "End of File!"
8. If there are more files to send, repeat steps 2 through 7 for each file.
9. When no more files remain to be sent, the two Kermits say good-bye to each other.

Two telephones cannot communicate with each other by themselves. They need a pair of people to operate them, and conversations will not be successful unless the two people follow the same set of rules. Likewise, two computers cannot transmit data using the Kermit protocol unless a Kermit program is running on *each* computer at the same time. What

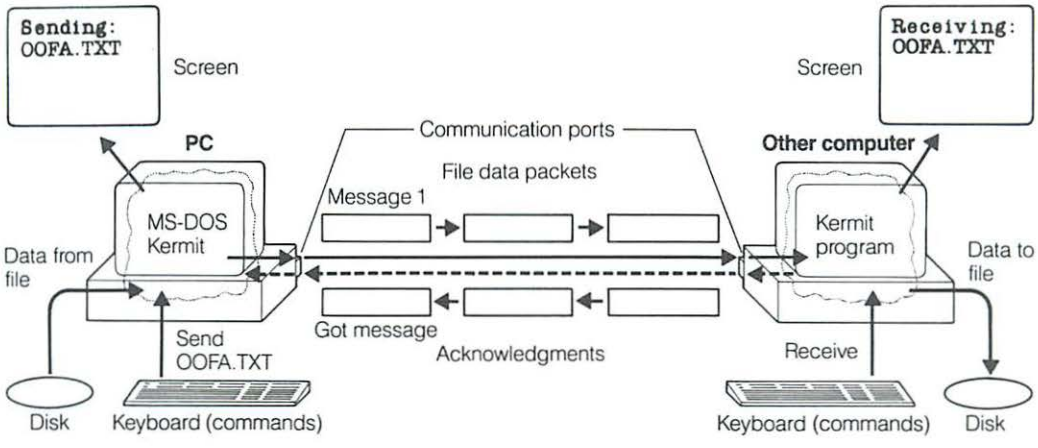


Figure 1-3 Kermit to Kermit

Chapter 2

Getting Started

The MS-DOS Kermit diskette contains the executable Kermit program, `KERMIT.EXE`; a summary of MS-DOS Kermit commands and functions, `MSKERM.HLP`; a "beware file," `MSKERM.BWR`, listing known limitations and bugs; and several utilities and sample files. (See Appendix I for a description of these files.)

To use the MS-DOS Kermit program, you must have:

- A PC: An IBM PC, PC*jr*, Portable PC, PC/XT, PC/AT, PC Convertible, PS/2, or any true IBM compatible such as the Compaq, DEC VAXmate or DECstation, or near-compatibles like the DG/1.[3]
- MS-DOS or PC-DOS 2.0 or later.
- A 3.5-inch or 5.25-inch diskette drive.
- The Kermit software on a diskette.[4]
- A blank diskette or a hard disk drive with at least 360K[5] of available space.

---

[3]Special versions of MS-DOS Kermit are also available for non-IBM-compatible PCs, including the DEC Rainbow, Victor 9000, HP-150, and many others.

[4]The MS-DOS Kermit diskette is included with this book.

[5]K means kilo, or one thousand. 360K is approximately 360,000 characters.

- At least 128K of available memory. Kermit will use more if it is available.
- At least one of the following communication devices:

  A serial port (asynchronous adapter) installed in your PC and, if you will be making data calls over a telephone, an external modem.

  An internal modem.

  A network adapter with vendor-provided driver software.
- A cable that connects either:

  Your PC's serial port to an external modem (a *modem cable*).

  Your PC's serial port to another computer (a *null modem cable*).

  Your internal modem to the telephone line (a *telephone cable*).

  Your network adapter to the network (a *network cable*).
- For Tektronix graphics terminal emulation, any of the common graphics adapters (CGA, EGA, VGA, Hercules, AT&T/Olivetti, and so forth).

Installation

MS-DOS Kermit may be run from a diskette or a hard disk. In either case, the very first thing you should do is make a working copy of the Kermit diskette from the original, and then put the original away for safekeeping. Since Kermit programs are not copy-protected, you can make as many copies as you like, give them away to your friends and colleagues, and contribute the program to your local user group for further distribution.

To install Kermit, follow the steps that apply to your PC's configuration.

Single Diskette Systems

If your PC has only one diskette drive, follow these steps to copy your Kermit diskette. `A>` is the DOS[6] prompt that tells you DOS is ready for you to give commands to it. You type only the commands that are underlined, and you must terminate each command by pressing the Enter key.

[6]DOS is short for MS-DOS or PC-DOS.

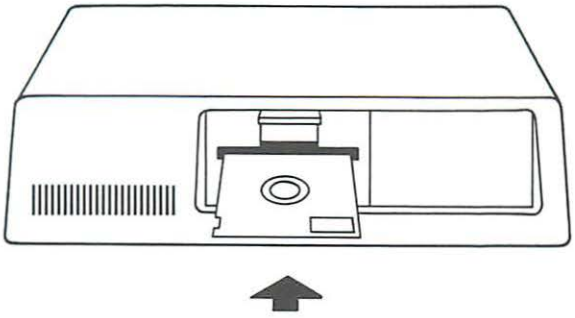


Figure 2-1 Installing Kermit in a Single-Diskette PC

1. Make sure your DOS diskette is in drive A.
2. Start up your PC (if it isn't already started). You should see the DOS `A>` prompt.
3. Make sure the write-protect notch on the Kermit diskette is covered so that you don't accidentally write over it. This is the *source diskette*.
4. Get a blank diskette and make sure the write-protect notch is uncovered so that you can copy the Kermit files to it. This is the *target diskette*.
5. Type `diskcopy a: a:`, and then press the Enter key.
6. A message will appear asking you to insert the source diskette into drive A. Remove the DOS diskette and put it safely out of reach. Put the original Kermit diskette into drive A (see Figure 2-1), and then press the Enter key.
7. After the DISKCOPY command has read as much of the source diskette as it can fit into its memory, it will ask you to remove the source diskette (original Kermit disk) from drive A and insert the target diskette (blank diskette). Do this, and then press the Enter key.
8. Repeat steps 6 and 7 until the Kermit diskette is copied. A message will ask you if you want to copy another diskette. Press N for NO.
9. Remove the new Kermit diskette, and then label it.
10. Put your new Kermit diskette into drive A.
11. At the DOS `A>` prompt, type `kermit`, and then press the Enter key. You should see the `MS-Kermit>` prompt. If not, review the previous steps.
12. At the `MS-Kermit>` prompt, type `exit`, and then press the Enter key to exit from the Kermit program.

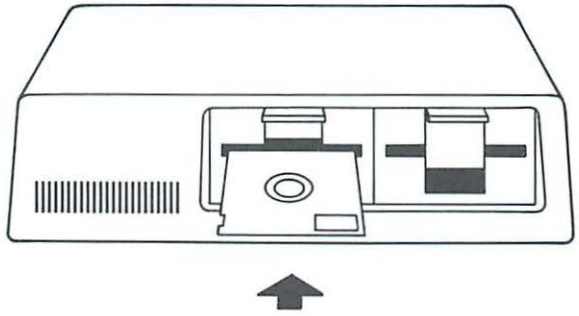


Figure 2-2 Installing Kermit in a Dual Diskette PC

Dual Diskette Drive Systems

On a system with two diskette drives, follow these steps. `A>` is the DOS prompt that tells you DOS is ready for you to give commands to it. You type only the commands that are underlined. Remember, you must terminate each command by pressing the Enter key.

1. Make sure your DOS diskette is in drive A (the upper or left diskette drive).
2. Start up your PC (if it isn't already started). You should see the DOS `A>` prompt.
3. Make sure the write-protect notch on the Kermit diskette is covered so that you don't erase it by mistake. This is the *source diskette*.
4. Get a blank diskette and make sure the write-protect notch is uncovered so that you can copy the Kermit files to it. This is the *target diskette*.
5. Type `diskcopy a: b:`, and then press the Enter key.
6. A message will appear asking you to insert the source diskette into drive A and the target diskette into drive B (lower or right diskette drive). Remove the DOS diskette, and put it safely out of reach. Put the original Kermit diskette into drive A and the blank diskette into drive B (Figure 2-2), and then press the Enter key.
7. When the diskette is copied, a message will ask you if you want to copy another diskette. Press N for NO.
8. Remove the new Kermit diskette, and then label it. Remove the original Kermit source diskette.
9. Put your new Kermit diskette back into drive B and the DOS diskette into drive A.
10. At the DOS `A>` prompt, type `b:kermit`, and then press the Enter key. You should see the `MS-Kermit>` prompt (if not, review the previous steps). To return to DOS, type `exit` and then press Enter.

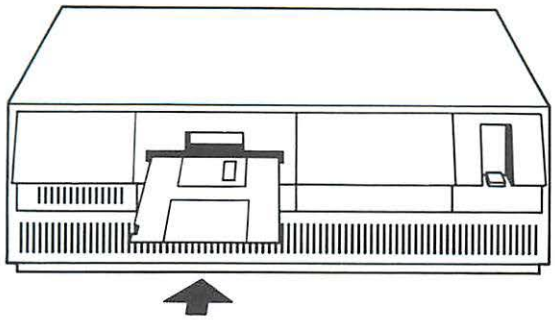


Figure 2-3 Installing Kermit in a PC with a Hard Disk Drive

Hard Disk Drive Systems

For easy access, you can copy KERMIT.EXE as well as all the other files on the original Kermit diskette to your hard disk. To copy all the files from your Kermit distribution diskette to your hard disk, you will need 360K available space on your hard disk. To see if you have enough space, give the following DOS command to C>, the hard disk prompt:

```
C>dir | find "bytes free"
```

Follow these instructions. You type what is underlined. Remember, you must terminate each command by pressing the Enter key.

1. Start up your PC (if it isn't already started). You should see the DOS prompt, which is usually C>.
2. Make sure the write-protect notch on the Kermit diskette is covered so that you don't destroy your Kermit files inadvertently.
3. Put the Kermit diskette into drive A (upper or left diskette drive). (See Figure 2-3.)
4. Make a directory on your hard disk to hold the Kermit files, and then make it your default directory. In the following example, we assume that your hard disk is disk C. The DOS prompt is C>, and the commands that follow it are the ones you should type. Terminate each command by pressing the Enter key:

   ```
   C>mkdir \kermit
   C>cd \kermit
   ```

5. At the DOS C> prompt, type:

   ```
   C>copy a:*.*
   ```

   and then press the Enter key.

6. The files are now in the KERMIT directory on your hard disk.

7. Remove the Kermit source disk from drive A.

8. At the DOS C> prompt, type:

   ```
   C>kermit
   ```

   Remember to press the Enter key after you type kermit. You should see the MS-Kermit> prompt. If not, review the previous steps.

9. At the MS-Kermit> prompt, type exit to quit the Kermit program and return to DOS.

Making Sure DOS Can Find Kermit

You should store the executable Kermit program, KERMIT.EXE, on your disk in a directory that is in your DOS search path. This will let you run the program simply by typing the word kermit, no matter what your current disk and directory are. The other files on the original Kermit diskette are best stored in the same directory on your hard disk. In the hard disk example above, all the Kermit files were placed in the directory \KERMIT.

To find out what your DOS search path is, type the command path at the DOS prompt. The response will be either:

```
No Path
```

or something like:

```
C:\;C\BIN;A:\;B:\
```

In the second case, the devices and directories are shown separated by semicolons. If you have no path, you should make one. If you do not know how to do this, see the section "Creating and Modifying Files" in Chapter 3 for an example. You can use a text editor to add the following line to your AUTOEXEC.BAT file, wherever that file is stored:

```
PATH=C:\;C:\KERMIT
```

This means that whenever you type a command that is not built into DOS, DOS should look on the C disk, first in the top-level directory and then in the \KERMIT directory, for a program of that name and run it. If you have put the KERMIT.EXE file in either of these directories on your C disk, you can run Kermit just by typing its name, kermit.

Chapter 3

Basics of MS-DOS

Before we proceed, let's review the basics of MS-DOS. I am sure you are anxious to begin using the Kermit program, but if you are not familiar with these concepts, this chapter is important. You should know about MS-DOS filenames, file formats, directories, devices, and commands.

MS-DOS means Microsoft Disk Operating System. This is the program that controls your PC and all its devices. In this book, the words DOS, MS-DOS, and PC-DOS will be used interchangeably.

DOS Filenames

A DOS filename has two parts, the *name* proper and the *type*. The two parts are separated by a period. For example, every DOS computer has a file called `COMMAND.COM`, which is the DOS command processor. The name is `COMMAND`, and the type is `COM`. Names can be up to eight characters long, and types can be up to three characters long. Legal characters include letters, digits, and certain punctuation marks. Letters are uppercase. If you type a lowercase letter in a filename, DOS converts it to uppercase. Some typical filenames are `AUTOEXEC.BAT`, `CONFIG.SYS`, and `KERMIT.EXE`. DOS filetypes, by convention, tell the purpose of the file, for example:

`.BAT` A DOS batch file, containing DOS commands to be run.

`.COM` A machine-language program to be run.

`.EXE` A machine-language program to be run.

`.DOC` Documentation, text to be printed or displayed on the screen.

`.TXT` Text to be printed or displayed on the screen.

`.HLP` A help file to be printed or displayed on the screen.

`.WKS` A Lotus 1-2-3 spreadsheet file.

`.DBF` A dBase database file.

`.PIF` A Microsoft Windows Program Information File.

`.BAS` A BASIC program.

`.C` A C-language source program.

There is no standard for filetypes, except that DOS will run only programs with filetypes of `.BAT`, `.EXE`, or `.COM`.

To display a file with any of the above filetypes on the screen, use the DOS `TYPE` command, for example:

```
C>type autoexec.bat
```

If the result is comprehensible and sensibly formatted, the file is probably a *text file*. On the other hand, if the result is "garbage"—a lot of funny-looking characters and sound effects—it is probably a *binary file*, which is meant not for viewing by humans but for processing directly by the computer. For example, the binary file `COMMAND.COM` is a PC machine-language program.

DOS Directories

DOS lets you organize your files into separate areas called *directories*. If you think of your disk as a filing cabinet, a directory is like one of its drawers. If you were a compulsive filer and each of your file drawers had its own index, the analogy would be complete.

Directories allow you to collect related files together. Each file within a directory must have a unique name, but files that are in different directories can have identical names. A directory name in DOS looks similar to a filename, but it begins with a backslash (`\`) character, for example:

```
\PROGRAMS
```

An important notion in DOS is the *current directory*. You can think of this as the file drawer that's open. To refer to a file in your current directory, you just type its name. To

refer to a file in a different directory, you have to include the directory name. A backslash must separate the directory name from the filename:

```
\PROGRAMS\OOFA.C
```

The *top-level* directory of any disk is simply \. Directories beneath it are called *subdirectories*. In the example above, \PROGRAMS is a subdirectory of the top-level directory, and OOFA.C is a file in that subdirectory.

To find out what your current directory is, just type CD at the DOS prompt, and then press the Enter key:

```
C>cd                          (Type the CD command)
C:\                           (See the result)
C>                            (Next DOS prompt)
```

This shows that your current directory is the top-level directory on the C disk.

To change your current directory, use the DOS command CD, specify the directory you want, and then press the Enter key, for example:

```
C>cd \programs                (Change directory)
C>cd                          (Check it)
C:\PROGRAMS                   (See, it worked)
C>
```

You can set up your DOS system to include the current disk and directory name in your DOS prompt so that you can always tell where you are. To do this, give the following command:

```
C>prompt $p$g                 (Old prompt is C>)
C:\>                          (New prompt shows directory)
C:\>cd \programs              (Change directory)
C:\PROGRAMS>                  (Prompt shows new directory)
```

Subdirectories may have their own subdirectories. For example, the PROGRAMS subdirectory could contain a subdirectory called NEW:

```
C:\PROGRAMS\NEW
```

If it doesn't, you can create it. To create a subdirectory, use the DOS MKDIR command:

```
C>mkdir \programs\new
```

You can use the RMDIR command to remove a directory (but only if it has no files in it).

You can move directly to a subdirectory within another subdirectory by telling DOS the full subdirectory name:

```
C>cd \programs\new
C:\PROGRAMS\NEW>
```

If you omit the initial backslash, the directory name is taken to be a subdirectory of the current directory. This is called a *relative path*. For example, if your current directory is \PROGRAMS, you can refer to a file in its NEW subdirectory as:

```
NEW\MYFILE.TXT
```

Two special directory names are related to your current directory: . (one period) means the current directory, and .. (two periods) means the directory immediately "above" the current directory. You can use these names in the CD command and in filenames. For example, if you were in the subdirectory \NEW\ and you wanted to get back to \PROGRAMS, the directory above it:

```
C\PROGRAMS\NEW>cd ..
C\PROGRAMS>
```

DOS Devices

Just as each directory may contain many files, each disk device may contain many directories. DOS disks are identified by letters: A, B, C, etc. Usually the A and B disks are diskettes, and the C disk, if any, is a hard disk. The disk letter may be included in a file specification to refer to a file that is on a disk different from the one your current directory is on:

```
A:\PROGRAMS\OOFA.C
```

Each disk has its own current directory, and you may switch among disks and their current directories by just typing the disk letter followed by a colon:

```
C>a:                          (Switch to disk A)
A>                            (Current disk is now A)
```

DOS also has nondisk devices, including the printer, the communication port, and the console. These are referred to by a simple name – no colon (as in the disk device name) and no dot (as in a filename). For example, COM1 is communication port 1, PRN is the printer, and CON is the console. These device names may be used in DOS commands in place of filenames, sometimes with bizarre results. A particularly useful device is the *null device*, NUL. Output sent to this device disappears mysteriously but with no ill effect. For input, NUL acts like an empty file. These device names may differ on non-IBM-compatible DOS systems.

Running DOS Commands

DOS has two kinds of commands: *internal* and *external*. An internal command is built into DOS, and you can always run it by typing its name in response to the DOS prompt, for example, the TYPE command:

```
C>type oofa.c
```

An external command is a file with a filetype of .EXE, .COM, or .BAT. You can run it by simply typing its name (either with or without the filetype) *if* it is in the current directory on the current disk. For example, if the file is called OOFA.EXE, you can type:

```
C>oofa
```

If it is on a different disk or directory, you must include the disk and/or directory when you type the command:[7]

```
A>C:\programs\oofa
```

or if the disk and directory is in your current PATH, you can just type its name:

```
A>path                          (What's the current path?)
PATH=C:\;C:\PROGRAMS            (C:\PROGRAMS is in it)
A>oofa                          (so I can just type OOFA)
```

It is normal practice to have the files that came on your DOS diskette always mounted in your A drive if you don't have a hard disk. If you have a hard disk, you should have copied the files from your DOS diskette into a directory on the hard disk. Let's say you have copied them to the top-level directory on the C disk. To make sure all of your DOS commands are in your path, you should have a line like this in your AUTOEXEC.BAT file:

```
PATH=C:\
```

The AUTOEXEC.BAT file is executed automatically by DOS when you start your PC. It must be in the top-level directory of the disk that you start DOS from.

For a diskette-only system, for instance one in which you have the DOS diskette in drive A and your working diskette in drive B, the PATH statement might look like:

```
PATH=A:\;B:\
```

Your PATH can include a list of disks and directories that DOS will search when you type a command name:

```
PATH=C:\;C:\PROGRAMS;A:\;B:\MISC
```

DOS will search these areas from left to right. So if you have two files named OOFA.EXE, one in C:\PROGRAMS and one in B:\MISC, and you type oofa, DOS will run the version it finds first, that is, C:\PROGRAMS\OOFA.EXE.

As you can see, the PATH mechanism of DOS allows you to add new commands simply by storing programs or batch files in any directory in your DOS search path.

---

[7]DOS 3.0 or later. In earlier DOS versions, you cannot specify a directory name when running a program.

Commonly Used DOS Commands

The most commonly used DOS commands are for *file management*: displaying, printing, copying, renaming, and deleting files. DOS commands like TYPE and DIR, which display text on your screen, can be controlled using the following *control characters*. Hold down the Ctrl key, and then type the indicated letter, but do *not* press the Enter key:

`Ctrl-C` Interrupts the command and returns you to the DOS prompt.

`Ctrl-S` Stops the display on the screen so that you can read it.

`Ctrl-Q` Resumes the screen display.

Here is a brief summary of file-related DOS commands. You must press the Enter key when you are finished typing the command and are ready for it to be executed. Refer to your DOS manual for details.

TYPE *filename*

Displays the file on your screen, for example:

```
C>type oofa.hlp
```

Try this: TYPE a long file, such as `KERMIT.HLP` on your Kermit disk. Use `Ctrl-S` (hold down the Ctrl key and press the S key) to stop the display and `Ctrl-Q` (hold down the Ctrl key and press the Q key) to resume the display. Use `Ctrl-C` (hold down the Ctrl key and press the C key) to cancel the TYPE command.

DIR

Lists the names, sizes, and creation dates of all the files in the current directory. If you include a disk letter (and the colon), directory name, or filename, the files in the specified area, or with the specified name, will be listed:

```
C>dir
C>dir a:
C>dir a:\programs
```

CHKDSK

Tells how much space is left on the current disk. If you include a disk letter (and the colon), the specified disk will be checked:

```
C>chkdsk c:
```

PRINT *filename*

Prints the file on your printer, if you have one:

```
C>print oofa.c
```

COPY *filename1 filename2*

Creates the second file, which is a copy of the first file. The files may be on different devices and directories:

```
C>copy oofa.c a:\programs\new
```

If *filename2* is omitted, or is specified as . (period), then the new files will be created on the current disk and directory with the same names as the original files:

```
C>copy a:*.*
```

REN *filename1 filename2*

Changes the name of *filename1* to *filename2*:

```
C>ren oofa.c serious.c
```

This command cannot be used to move a file to another device or directory.

DEL *filename*

Deletes (removes, erases) the file:

```
C>del oofa.exe
```

Wildcards

Certain DOS commands, like DIR, COPY, PRINT, and DEL, can operate on more than one file at a time. To specify a group of files, you can use *wildcard* characters in the filename (but not a directory or device name). DOS recognizes the following two wildcard characters:

`*` (asterisk) Matches all characters from the current position to the end of the current field (filename or filetype).

`?` (question mark) Matches a single character in the current position.

So `O*.HLP` would match any file (like `OOFA.HLP`) whose name started with the letter `O` and that had a filetype of `.HLP`. The names could be of different lengths, like `O.HLP`, `OK.HLP`, `OIL.HLP`, and `OOMPAH.HLP`. If all of these files existed in your current directory, the command:

```
C>del o*.hlp
```

would delete all of them. The specification `*.*` matches all the files in the current directory. If you tell DOS to:

```
C>del *.*
```

DOS will respond:

```
Are you sure (Y/N)?
```

This gives you a chance to change your mind. Combining wildcard characters can result in more specific selections; for example:

```
O?F?*.?O?
```

would match all files whose names start with the letter O, followed by any character, followed by the letter F, followed by one or more additional characters, with a filetype exactly three letters long whose middle letter was O. This would match files with names like OOFA.DOC and OOFAS.COM. Although this method may seem cumbersome, it is the only method DOS provides for identifying many files all at once. Once you become used to wildcards, they can save you a lot of typing and time.

Creating and Modifying Files

To create a new file, or to modify an existing one, you should use a *text editor* or *word processor*. DOS includes a very simple text editor called EDLIN, which is described in the DOS manual. There are many other text editors that are proprietary programs and that include a large selection of powerful features.

Let's use EDLIN to add two lines to your AUTOEXEC.BAT file: one to set your DOS prompt to show your current disk and directory, and another to set up your DOS path. For simplicity, we will add them to the end of the file, but you could also put them in the beginning or the middle of a file:

```
C>edlin \autoexec.bat
End of input file
*i#                                  (Insert lines at end)
      12: PATH C:\;C:\KERMIT         (Add PATH command)
      13: PROMPT $P$G                (Add PROMPT command)
      14: Ctrl-C                     (Hold down the Ctrl key and)
                                     (press the letter C)
                                     (to leave insert mode)
*e                                   (Exit and save the file)
C>
```

This will work even if you don't already have an AUTOEXEC.BAT file. EDLIN will create one for you. The PATH command shown above is only a sample. Replace it with one appropriate to the organization of your disks and directories.

Chapter 4

Cables, Connectors, and Modems

Your PC cannot communicate with another computer until you have established a physical connection (see Figure 4-1). If you already have a data connection to the other computer, skip ahead to Chapter 5 to check that it is working properly.

There are several methods you can use to connect your PC to the outside world, depending on the communication hardware you have. If your PC does not have any kind of communication hardware, you must decide what to buy.

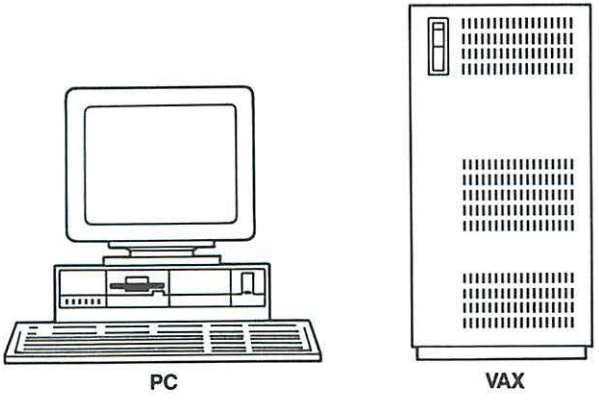


Figure 4-1 A Connection Waiting to Happen

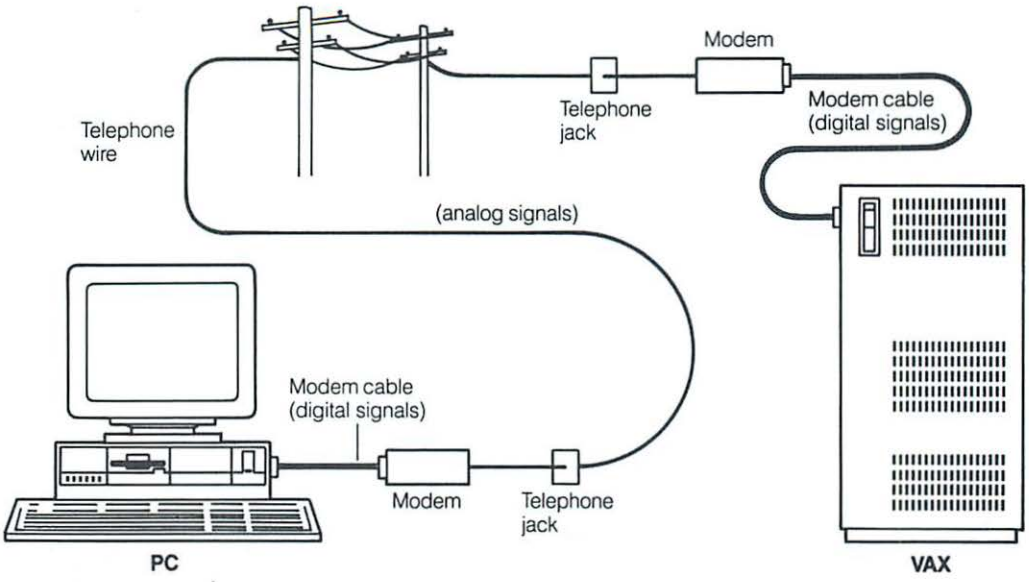


Figure 4-2 Computers Connected by Telephone

Just as there are many kinds of computers you can connect to, there are many ways to make the physical connection. Once you establish the best way to set up your PC connection, however, you will probably not need to alter the connection. So, think of this chapter as a one-shot deal, and it should seem less painful.

Modems

If you will be using your PC to make data connections over the telephone system, you will need a *modem*, which is a device that allows computer data to be transmitted over ordinary telephone lines by converting between digital computer signals and analog telephone signals. To communicate in this manner, *both* computers must have a modem—one to convert to analog signals so that the data can travel over telephone wires and one to convert back to digital signals so that the computer can understand the data. (See Figure 4-2.)

Unless both communicating computers belong to you, you have to be concerned only with the modem for your own PC. Bulletin boards and other dialup services already have modems on their end that are set up for incoming calls.

You can choose from two types of modems: *external* and *internal*. An external modem is a separate unit that lies between your PC's serial port and the telephone wall jack. An internal modem connects directly to the telephone wall jack.

Although an internal modem is less expensive and does not take up any space on your desk, it is also less flexible. An external modem can be used on different types of computers and is easily mobile. An internal modem can usually be used on only one type of computer and is physically installed inside it. Kermit works with any external modem, but Kermit does not necessarily support all makes and models of internal modems. Therefore, if you must buy a modem, the external type is more widely recommended.

Locating and Identifying Your Communication Device

Before you can use your PC for communications, you must have an *internal modem* or a *serial port*. These are circuit boards installed in your PC that have a connector for a cable. Look at the back of your PC. You will see a number of connectors for various devices. Unfortunately, these connectors are rarely labeled. Finding your communication device might therefore require a little detective work.

Internal Modems

If your PC has an internal modem, you will see a modular telephone jack somewhere on the back of your PC. This telephone jack allows the modem to be connected to your telephone wall jack using a modular phone cable (available in hardware stores and supermarkets).

The internal modem can be used only to communicate with another modem. It cannot be connected to another computer's serial port or to other data communication equipment like terminal servers and multiplexers.

Serial Ports

The serial port, also called the *asynchronous adapter* or *communication port*, can be used to communicate directly with another nearby computer, or it can be used with an external modem to communicate with more distant computers or services that can be dialed by telephone. It can also be used with other kinds of data communication equipment, including terminal servers, multiplexers, and PBX data phones.

A PC serial port connector is shaped like an elongated capital letter D. For this reason, these connectors are called *D-connectors*. D-connectors come in two sizes. One size has 25 holes, or pins, in two rows: 13 in one row and 12 in the other. This size is called DB-25. A male DB-25 connector may have fewer than 25 pins, but it always has 25 positions for pins. The female DB-25 always has 25 holes. The other size is called DB-9. It has 9 holes or pins in two rows: 5 in one row and 4 in the other.

D-connectors also come in two "genders," female and male. A female connector has holes, and a male connector has pins. This is similar to an electrical outlet (female) and plug (male).

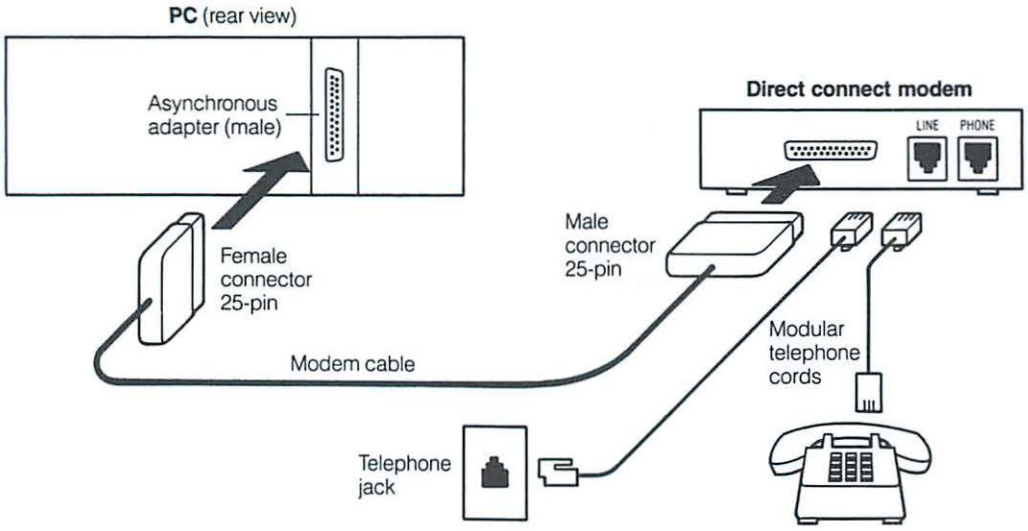


Figure 4-3 A Direct Modem Connection

The PC serial port uses a *male* DB-25 or DB-9 connector. Female connectors are used for other devices like printers and cannot be used for communications.

In general, you will find DB-25 serial port connectors on IBM PCs and PC/XTs, and DB-9 connectors on PC/ATs. On the PC/AT, the DB-9 is on the same card (called the Serial/Parallel Adapter) with a DB-25 female printer port.

The PS/2 has a DB-25 as its first communication port, but additional ports, if present, have DB-9 connectors.

Serial Port Cables

To communicate, you must plug a data cable into your serial port. The end that you attach to your port must be the same size and shape, DB-25 or DB-9, as the port itself but of the opposite gender. Since the port connector is male, the cable connector must be female.

The connector on the other end of your cable depends on what you will be plugging it into. In most cases, this will be a data communication device such as an external modem. Almost all such devices have *female* DB-25 connectors, so the far end of your cable would need a *male* DB-25 connector.

A cable that connects your PC to a data communication device is called a *modem cable*. Modem cables for PCs, PC/ATs, and PS/2s are readily available in computer stores and supply catalogs.

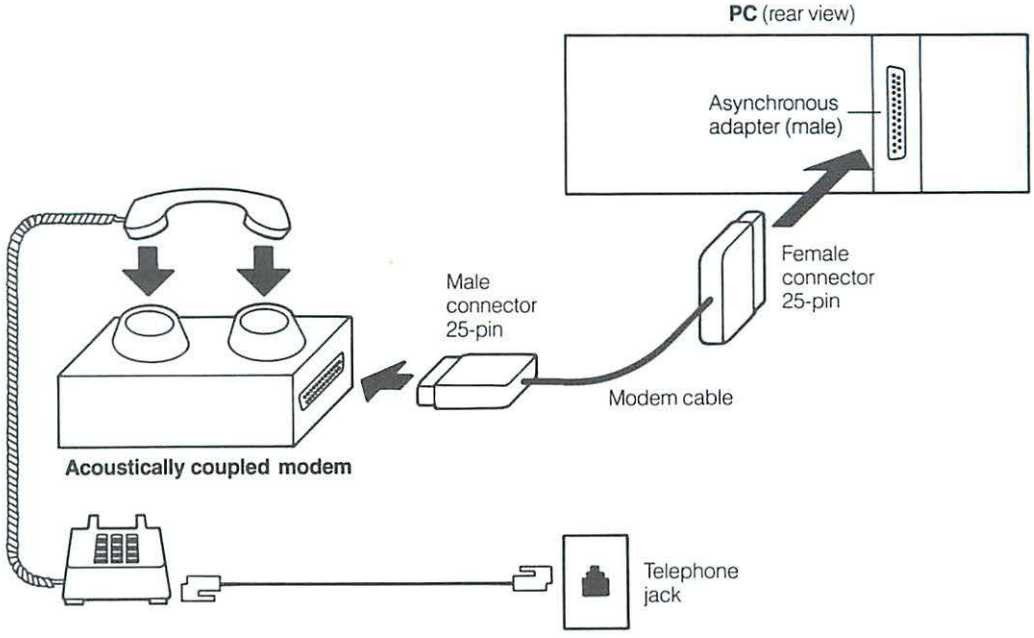


Figure 4-4 An Acoustically Coupled Modem Connection

If you have a serial port but do not have a cable for it, you should buy a modem cable. Be sure to specify the size and gender of both the serial port connector and the connector on the device you will be connecting your PC to. If you will be connecting your PC directly to another computer, you will also need an adapter called a *modem eliminator*, which is explained later.

Connecting Your PC to an External Modem

External modems are probably the most common way for PC users to connect with other computers. Most external modems have a 25-pin female connector, which must be connected to your PC's serial port with a modem cable. Be sure you are using a modem cable, *not* a printer cable or a *null modem* cable. Be sure also to make a firm connection at both ends, even if that means using the screws.

If your modem is a direct-connect model, you must also use a modular telephone cable to connect it to the telephone jack where your telephone normally plugs in, as shown in Figure 4-3. Most direct-connect modems also let you plug your telephone into the modem so that the phone can be used for voice calls without disconnecting the modem.

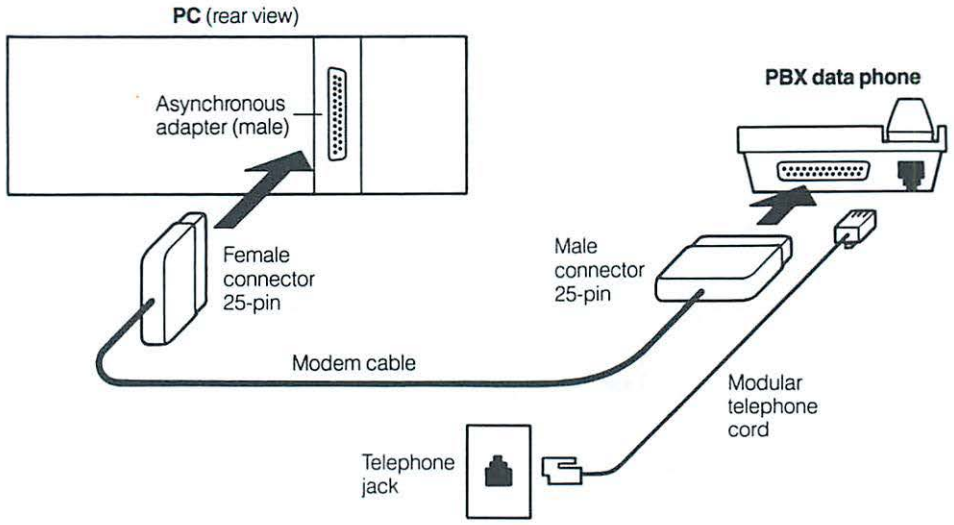


Figure 4-5 Connection to a PBX

If your modem has an acoustic coupler (a rare item today), you won't need a modular telephone cable. Just push the telephone handset's earpiece and mouthpiece firmly into the modem's rubber cups, as shown in Figure 4-4.

Here is the normal procedure for connecting your PC to a direct-connect external modem (see your modem manual for more specific details):

1. If your telephone does not have a modular jack, buy a telephone that has one, and convert your wall jack to a modular one. The parts are available in any hardware store.
2. If the modem has a power switch, turn it off. Connect the power cable and transformer, if any, to the modem and to an electrical outlet.
3. Connect the modem to the PC using your modem cable.
4. Disconnect the modular telephone cable from your telephone, and plug it into the modular phone jack on the modem. If there are two such jacks, use the one marked LINE or TO LINE.
5. If your modem has two modular jacks, use the short modular telephone cable that came with the modem to connect it to the telephone. One end goes into the modem jack marked PHONE or TELSET, and the other end goes into the empty jack on your telephone from the previous step.

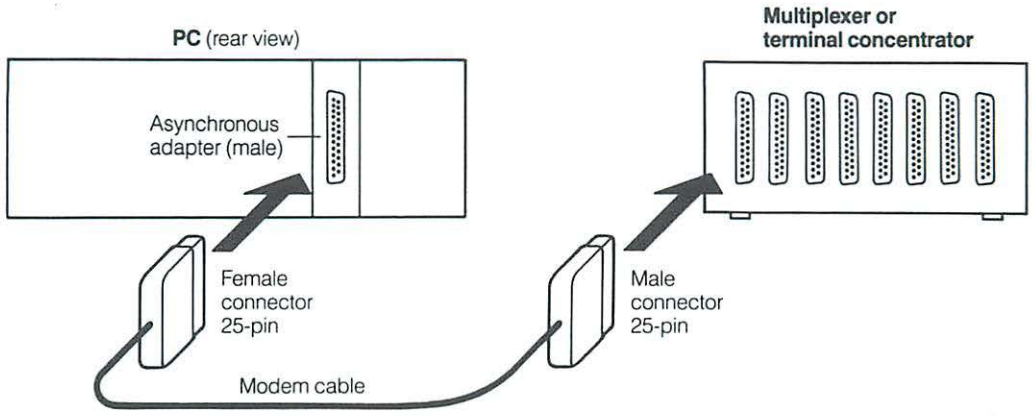


Figure 4-6 Connection to a Multiplexer or Terminal Server

Connecting the modem to your phone is not necessary for data communication. It just lets you use your phone for regular voice calls without having to unplug all the cables you have just connected. When the modem is turned off, you can use your phone normally.

The danger here is if other people have access to the telephone line you are using. For example, if you are dialing Dow Jones to find out the progress of your new stock and someone picks up an extension, your data connection will be lost. The connection can also break if you have "call waiting," so be sure to inform potential callers in advance when you want to use the phone for data or (if your telephone service provides this option) turn off call waiting for the duration of your modem call.

PBX Data Lines and Other Communications Equipment

Many large organizations have their own internal telephone systems called PBXs (Private Branch Exchanges), which can accommodate both voice and data calls on the same telephone line simultaneously. Connection to a PBX telephone is very similar to connecting to an external modem. Typically, the telephone will have a 25-pin female data connector just like an external modem. However, in this case, you connect your PC directly to the telephone's data connector using a modem cable, as shown in Figure 4-5, but you should leave the telephone cables alone. (See your PBX telephone manual for further information.)

Other data communication devices, including port contention units, terminal servers, and multiplexers, are generally just boxes that have one or more modemlike 25-pin female connectors, as in Figure 4-6. In this case, you can connect your PC directly to an unoccupied connector on the data communications device using a modem cable. (See the manual for the specific data communications device for details.)

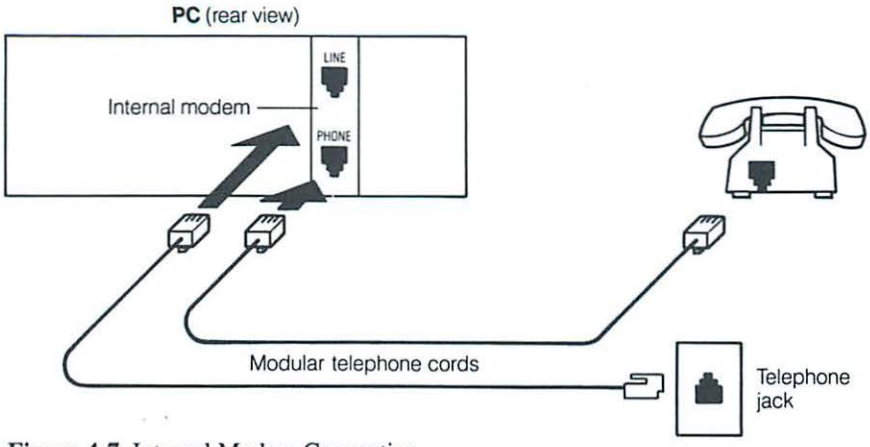


Figure 4-7 Internal Modem Connection

Connecting an Internal Modem to the Telephone Line

An internal modem can be used only with a modular telephone. Connect your internal modem to your telephone's wall jack. To do this, unplug the modular cable from your telephone and plug it into your internal modem's modular jack on the back of your PC. If there are two jacks on your internal modem, connect the one marked LINE to the wall jack and the one marked PHONE to your telephone using a short modular telephone cable, as in Figure 4-7.

Connecting Your PC Directly to Another Computer

If you will be connecting your PC directly to another nearby computer (one, that is, within about 50 feet or 15 meters), your PC must have a serial port. (See Figure 4-8.) Certain signals must be crossed when connecting a computer to another computer instead of to a modem. The recommended method is to use a regular modem cable, just as you would use with an external modem, in conjunction with a *null modem adapter* or *modem eliminator*. This is a small, squarish, solid adapter plug, with a 25-position connector on each end. Internally, the pins and holes are interconnected in mysterious ways you needn't bother about. Modem eliminators are available in computer stores and catalogs, and come in three possible *gender* configurations: female-male, female-female, and male-male.

Your choice of null modem depends on which kind of connector the other computer has:

- 25-pin male connector: Connect your modem cable to the other computer through a female-female null modem.

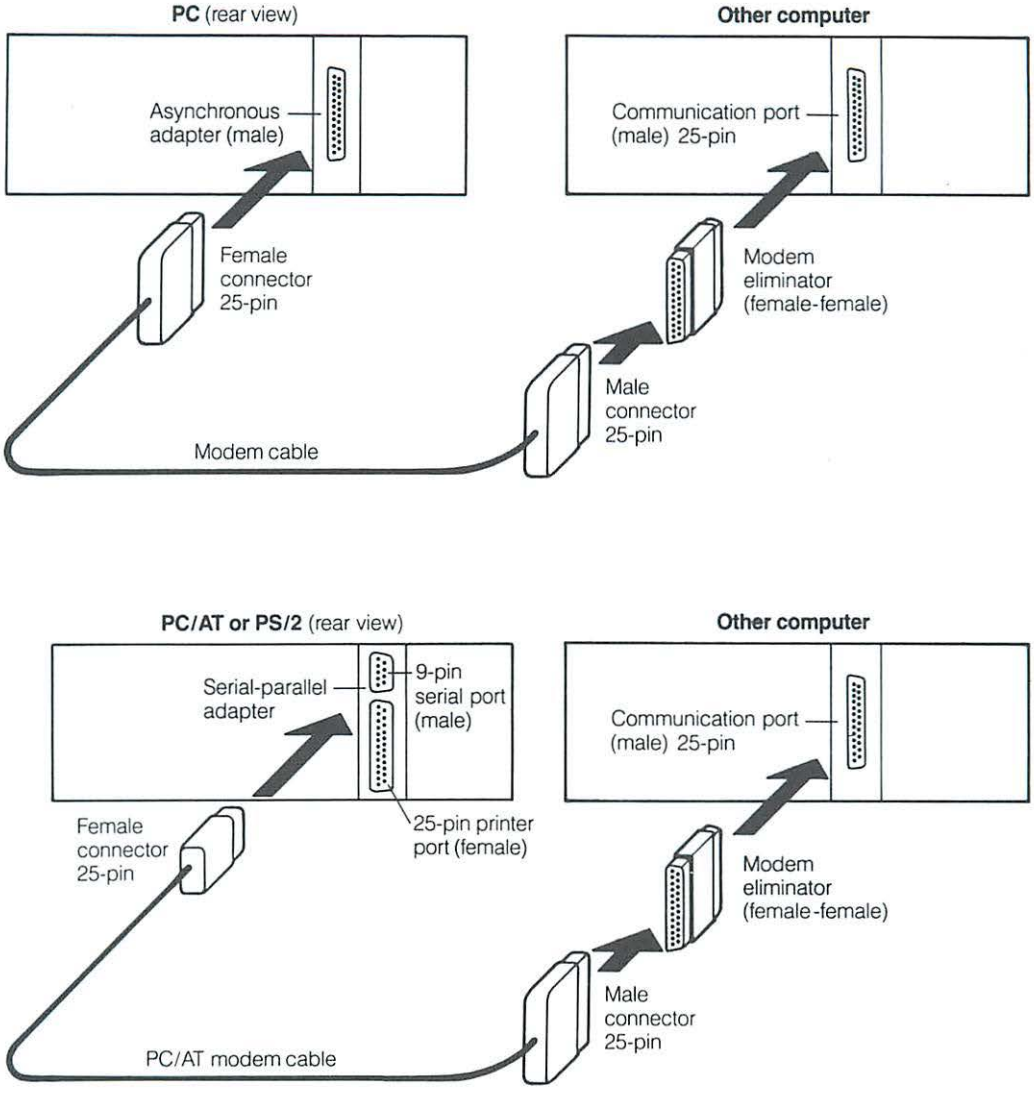


Figure 4-8 Direct Connection, Computer to Computer

- 25-hole female connector: (Make sure this is not a parallel printer port.) Connect your modem cable to the other computer through a female-male null modem.
- Anything else at all: Use the modem cable supplied with the other computer to connect it to your PC's modem cable, with a female-female null modem in between.

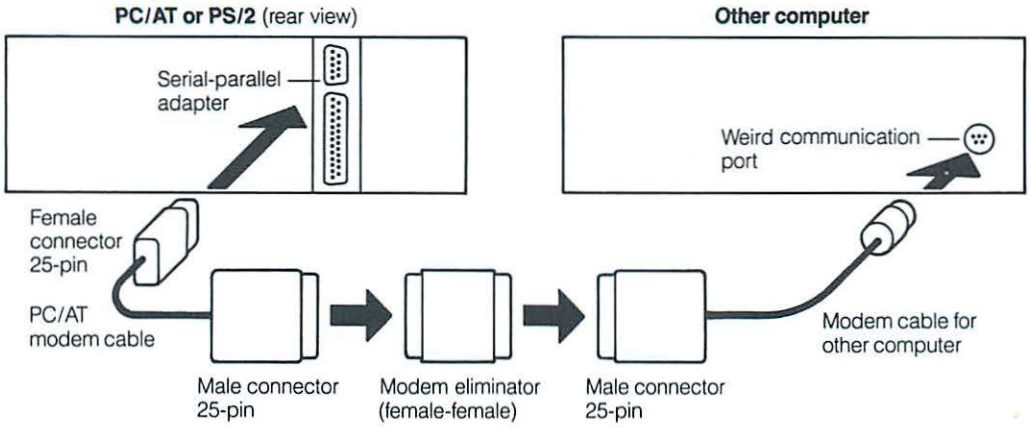


Figure 4-8 ***(Cont.)*** Direct Connection, Computer to Computer

For example, suppose you want to connect a PC/AT with a Macintosh II. Neither of these systems has a standard 25-pin connector. The PC/AT has a 9-pin connector, and the Macintosh has an 8-pin connector. You will probably not be able to find a null modem cable in any store that will connect these two computers. But it's easy to find *modem* cables for both the PC/AT and the Macintosh. Now all you need is a female-female modem eliminator between the two modem cables, and you're all set.

Chapter 5

Testing the Connection

If you have never used your communication hardware, cable, or modem before, you should test it now. Otherwise you might blame the innocent MS-DOS Kermit program for any failure to communicate.

Direct Connections

Read this section if you have two computers connected with only a cable—no modems involved. There are two cases: PC to PC and PC to host computer.

PC to PC

The two PCs should have a null modem connection (see Figure 5-1). You can perform this test by running back and forth between the two PCs, or you can have a friend handle the PC at the other end.

To test the connection, run Kermit on both computers, setting the programs to the same transmission speed and connecting them with the proper Kermit command. When you type characters (letters, numbers, and keyboard symbols) on one of the PC keyboards, they should appear on the other PC's screen. This should work in both directions.

Follow this example. For simplicity, let's assume you have your Kermit diskette in the A drive. In the example, the parts you type are underlined and should be terminated by pressing the Enter key, except where you see `Alt-X` (which means hold down the Alt key and press the X key).

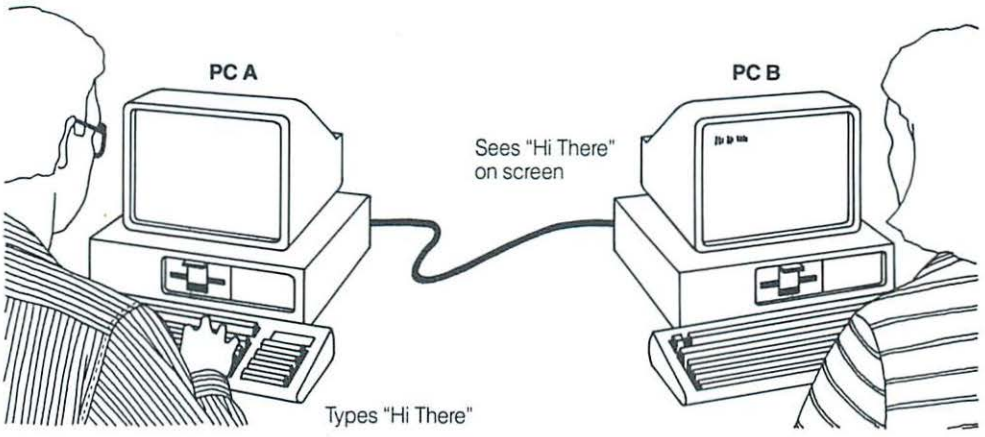


Figure 5-1 Testing a PC-to-PC Connection

PC Number One

```
A>kermit
MS-Kermit>set port 1
MS-Kermit>set speed 9600
MS-Kermit>set local on
MS-Kermit>set term newline on
MS-Kermit>connect
Hi, can you read this?
Yes, I can read it!
Alt-X
MS-Kermit>exit
```

PC Number Two

```
A>kermit
MS-Kermit>set port 1
MS-Kermit>set speed 9600
MS-Kermit>set local on
MS-Kermit>set term newline on
MS-Kermit>connect
Hi, can you read this?
Yes, I can read it!
Alt-X
MS-Kermit>exit
```

If the Test Failed

- Is your cable securely attached at all points?
- Are you sure you have a null modem cable or a cable with a null modem adapter? If you have a null modem adapter, try removing it. If you don't have one, buy one.
- Are you sure the connector on the back of the PC that your cable is plugged into is a communication port and not a printer port?
- Are you sure your PC *has* a communication port? If it doesn't, you'll have to go to the computer store and get one.
- Are you sure the cable is connected to communication port number 1 (COM1) and not, for example, to COM2? Try repeating the example above, but use SET PORT 2 rather than SET PORT 1 (on one PC at a time).

If none of these questions produces an explanation of the problem, then several possibilities remain:

- Your serial port is broken. Read the technical manual for your communication device, and then run any available diagnosic tests. If it fails, it must be fixed or replaced.
- Your serial port or internal modem is misconfigured. Read the installation manual that came with it. For example, you might have installed it as COM1 when there already was another COM1. If you have two ports, one of them should be configured as COM1 and the other as COM2.
- Your cable is defective. It must be fixed or replaced. Read the remainder of this chapter for some cable testing procedures.
- Some other software is interfering with Kermit. Look in your `CONFIG.SYS` and `AUTOEXEC.BAT` files for mouse drivers, alarm clock drivers, pop-up utilities, keyboard drivers, terminate-and-stay-resident (TSR) programs, and so on. Remove them all. Repeat the Kermit test. If it works, start putting back your other software, one program at a time, until Kermit stops working again. Then you know which program is the culprit. Don't run the offending program at the same time as Kermit.

Problems can occur in any combination, particularly in an old or inherited PC, and different (or the same!) problems can occur on both PCs at once—which means that finding and fixing one problem won't necessarily fix your communications. Be patient, keep trying.

Pinpointing the Problem

You can't communicate at all. Is the problem Kermit, the PC, the port, the connector, the cable, the other computer, or something else? There is an easy way to narrow this down: the *loopback connector*. A loopback connector plugs into your serial port. It has no cable. Inside the connector, the receive and transmit signals are connected. This means that whenever the PC sends a character, it immediately receives the same character. Loopback connectors are available from many computer supply catalogs and computer stores. If you can find one, connect it to your port, and then run the following test:

```
A>kermit                          (Run MS-DOS Kermit)
MS-Kermit>set port 1              (Select the right port)
MS-Kermit>set local off           (Let loopback do the echoing)
MS-Kermit>connect                 (Begin terminal emulation)
abcdefghijklmnopqrstuvwxyz        (Type some characters)
Alt-X                             (Escape back to the PC)
MS-Kermit>exit                    (Exit to DOS)
A>
```

After giving the CONNECT command, type some characters, such as the alphabet shown in the example. If you see these characters on your screen, Kermit is working, your PC is working, and the port is working; therefore, the problem must be in the cable or the other computer. Move the loopback connector to the other PC and perform the same test. If it works, the problem must be in the cable or modem eliminator.

If a PC fails the loopback test, the most likely causes are:

- Kermit is trying to use a communication port different from the one that the loopback connector is on. Use Kermit's SET PORT command to try different ports, or move the loopback connector. If you still can't make it work, then:
- There is no serial port. You have your loopback connector plugged into something else. Remove the loopback connector and go out and buy a serial port.
- The serial port is misconfigured. If you have one serial port, it should be configured as COM1. If you have two, they should be configured as COM1 and COM2. Read the installation instructions that came with your serial port.
- The serial port is broken. Get it fixed or buy a new one.
- The slot your serial port is plugged into is bad. Remove the cable from the port, take the cover off your PC, pull out the serial port card, and plug it into a different slot. If you don't have a spare slot, plug it (firmly) back into the same slot. Run the test again. If you still can't communicate, your PC might need repair.

To check the cable itself, plug one end into the PC, and attach the loopback connector to the other end. Repeat the test. If it works, the problem is in the modem eliminator, and our loopback connector is a problem eliminator.

PC to Host Computer

If you have a direct cable connection (with a null modem) between your PC and a multi-user host computer (for example, a VAX minicomputer with the VMS operating system; see Figure 5-2), you can check the cable by running the Kermit program on the PC, setting MS-DOS Kermit to the same speed as the other computer, and then connecting your PC to the host computer with the proper MS-DOS Kermit command.

```
A>kermit                          (Run MS-DOS Kermit)
MS-Kermit>set speed 9600          (Set the speed)
MS-Kermit>connect                 (Begin terminal emulation)
Welcome to the VAX/VMS system     (Press Enter to get greeting)
Username: xxxxxx                  (Type some characters)
Alt-X                             (Hold down Alt and press X)
```

What happened?

- You saw the greeting message of the multiuser computer; the cable is fine, the port is fine, Kermit is fine. Proceed to Chapter 6.
- If, after typing the CONNECT command, you saw nothing, the trouble may lie in the cable, port, or PC, as described in the previous section of this chapter. Go back and read it. Try the loopback test if you can. It is also possible that the host port is broken or misconfigured. You will have to check with the administrator of the host computer.
- If you saw something, but it was not recognizable, Kermit's communication parameters might need to be adjusted to what the host expects. But you don't know how to do this yet. Skip the rest of this chapter and read Chapters 6 and 7. Then set up Kermit correctly for your host and try again.

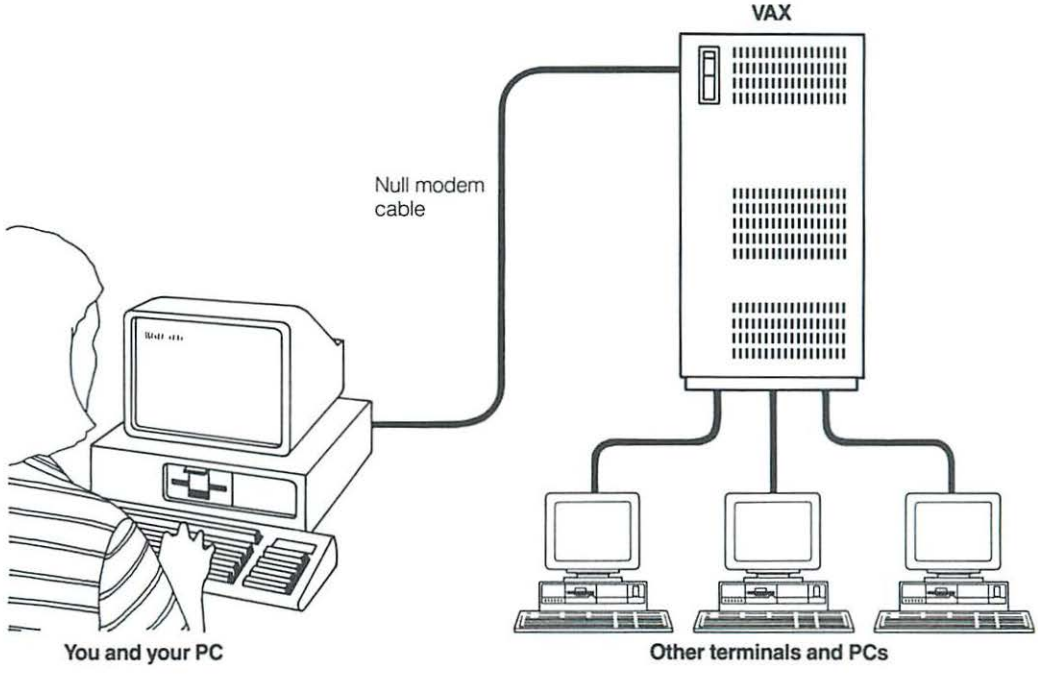


Figure 5-2 Testing the PC-to-Host Connection

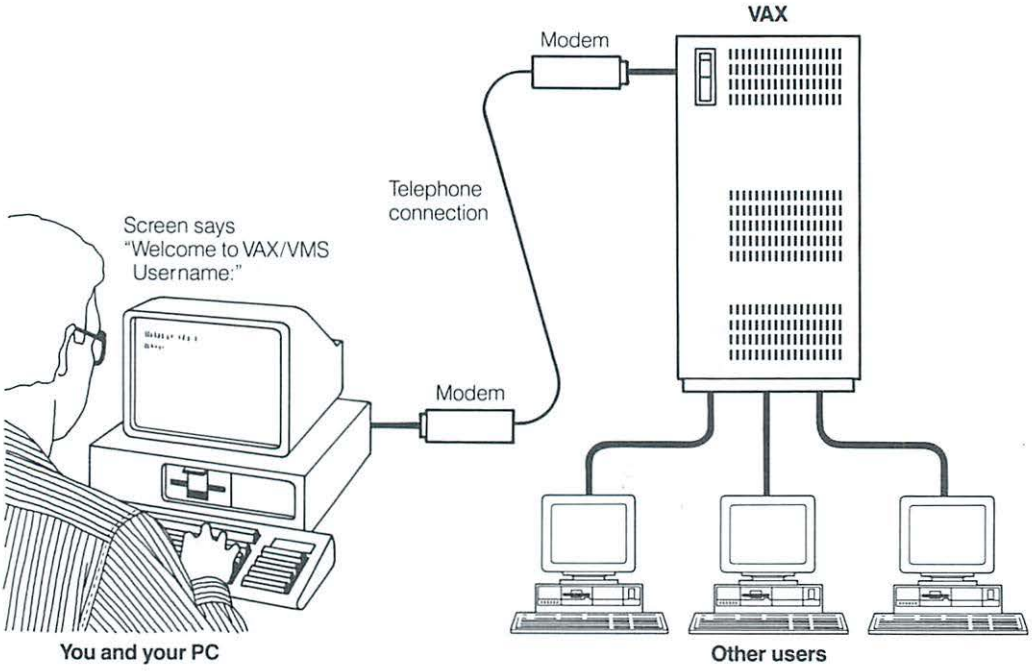


Figure 5-3 Testing a Modem Connection

Modem Connections

If you are using a modem, internal or external, your PC must make other connections before it can communicate with another computer. First, the PC must be able to talk to your modem, then the modem must be able to dial a telephone number and go across the telephone wires to talk to the other computer's modem, and finally the other modem must be able to talk to the other computer. (See Figure 5-3.)

This sounds more difficult than it is because normally you need to worry only about connecting your PC to your modem and instructing the modem to dial the number. Either the other modem will pick up and you will be connected or you will get an informative message like `BUSY` or `NO ANSWER`.

Modern modems include an autodialer that will dial the phone number for you. Though different modems work in different ways, the most popular method is the one used by Hayes Smartmodems and copied by the many other manufacturers that claim to make "Hayes-compatible" modems.

Hayes dialing commands are sent to the modem by the PC. (You do not press them on your telephone keypad!) The commands begin with the letters `AT`. If you type the com-

mand AT and then press the Enter key, the modem should respond with the message OK. If you do not see the OK message on your screen:

- If your modem has a power switch, check that it is turned on.
- Check that your cables are plugged in tightly.
- Try resetting the modem with the command ATZ.

The letters that follow the modem command AT tell the modem what to do; DT means Dial Touch-tone, and DP means Dial Pulse (rotary). After the modem dialing command ATDT or ATDP comes the telephone number. The modem command is not executed until you press the Enter key. For example:

```
ATDT7654321
```

This is the Hayes modem command for dialing the telephone number 765-4321 on a Touch-tone telephone.

In this book, dialing will always be illustrated by using Hayes commands since this method is so common. If you have a different type of autodial modem, substitute your own modem's dialing commands. If your modem does not have a built-in dialer, you must dial the telephone number manually, just as you would for a normal telephone call. Consult your modem manual for details.

Dialing Up an Information Service

Preliminary details to consider:

1. Is your modem Hayes compatible?	A. ❒ Yes	B. ❒ No
2. What is your modem's speed?	C. ❒ 1200	D. ❒ 2400
3. How does your phone dial?	E. ❒ Touch-tone	F. ❒ Pulse (Rotary)

Now let's try to call up the Digital Equipment Corporation Electronic Store. You don't have to buy anything; you don't even have to tell them who you are—"Just looking!" Here's what you do:

1. Make sure your modem and your PC are turned on[8] and your modem is connected as described in Chapter 4 (did you read it?).

[8]Some external modems receive their electrical power from the communication line and don't need power from an electrical outlet. Such modems might not have an on/off switch, so you won't need to turn them on. See your modem manual. Internal modems never need to be turned on.

2. At the DOS prompt, type `kermit`, and then press the Enter key:

```
C>                              (The DOS prompt)
C>kermit                        (Type KERMIT, press Enter key)
```

If you see the message `Bad command or file name`, you didn't install Kermit correctly. Go back to Chapter 2, check your installation, and then come back here.

3. You will see MS-DOS Kermit's greeting and prompt:

```
IBM-PC MS-Kermit: 3.0
Copyright (C) Trustees of Columbia University 1982, 1990
Type ? or HELP for help
MS-Kermit>                      (This is Kermit's prompt)
```

4. At this point, you can type any Kermit command. Let's begin by setting the transmission speed:

```
MS-Kermit>set speed 2400
```

If you checked box C above, type `set speed 1200` instead.

5. Now connect to the modem and type the dialing command: If you checked box B above, substitute your particular modem's dialing command or dial the number manually on your telephone. Otherwise, type only one of the ATD commands shown below, depending on the type of dialing your phone does. Be sure to dial the right phone number, or you might hear someone answer "Hello?" on your modem speaker.

```
MS-Kermit>connect                (Connect Kermit to your modem)
AT                               (Type AT, then press Enter)
OK                               (Modem should respond OK)
ATDT 1-800-234-1998              (Hayes with Touch-tone, Box E)
or
ATDP 1-800-234-1998              (Hayes with pulse dial, Box F)
```

If the modem doesn't respond with `OK`, make sure it's turned on and properly connected (see the section "Connecting Your PC to an External Modem" in Chapter 4). If it is, go back and read the section "PC to PC" earlier in this chapter.

6. Wait about 30 seconds. If the call is answered successfully, you should see one of the following messages (if you have a Hayes or Hayes-compatible modem):

```
CONNECT
CONNECT 1200
CONNECT 2400
```

If the message shows 1200 when you dialed at 2400, you must tell Kermit that the connection speed has changed behind its back, for example:

```
CONNECT 1200
Alt-X                            (Hold down Alt key and press X)
MS-Kermit>set speed 1200         (Change the speed to 1200)
MS-Kermit>connect                (Connect back to modem)
```

If the message was BUSY or NO ANSWER, wait a little while, and then go back to step 5 and try again.

7. Now press the Enter key and wait a few seconds. If you don't see anything, press the Enter key again. Repeat several times until you see a greeting, and then follow the directions.

Select the menu item "View a Demonstration of the Electronic Store." You will see a nice demonstration of MS-DOS Kermit's DEC VT terminal emulator. Just follow the instructions on the screen. Note that <RETURN> means you should press the Enter key, and CTRL_C means you should hold down the Ctrl key and press the letter C.

You can wait for the demonstration to finish, or you can leave it prematurely. In either case, you should terminate your session by hanging up, just as you would on a regular phone call. This concludes our test drive. Exit Kermit and return to DOS as follows:

```
Alt-X                          (Hold down the Alt key and press X)
MS-Kermit>hangup               (Get Kermit prompt and type HANGUP)
MS-Kermit>exit                 (Exit from Kermit)
C>                             (Back at the DOS prompt)
```

Modem Problems

If you have a Hayes or other autodial modem, but you never saw anything on your screen after you gave the CONNECT command, you must face all the possible problems listed in the section "PC to PC" at the beginning of this chapter. Use the methods described there to pinpoint the problem.

In addition, your modem may be turned off, not working, or not configured correctly. Read your modem manual. Try your modem on somebody else's PC to see if it works there. If it doesn't, the modem needs fixing or replacing.

You can also use an external modem to test the serial port. Most external modems have status lights, which include a receive (RD or RxD) light and a transmit (SD, TD, or TxD) light. After giving the Kermit CONNECT command, type some characters. Do you see the transmit light blinking? If not, something is wrong with your serial port or the cable to your modem.

Now let's find out more about MS-DOS Kermit's commands.

Chapter 6

Running MS-DOS Kermit

Now you have a working copy of the Kermit program for your PC and a tested data connection to another computer. So far, so good. Now we can discuss the MS-DOS Kermit software program itself.

There are two things you should find out whenever you plan to use a computer software program: how to start the program and how to exit from it. You already saw how to do both of these in previous sections, but let's go over it one more time.

Starting and Stopping the Kermit Program

To run the MS-DOS Kermit program on your PC, type `kermit` at the DOS prompt, and then press the Enter key:

```
C>kermit
IBM PC MS-Kermit V3.0
Copyright (C) Trustees of Columbia University 1982,1990
Type ? or HELP for help

MS-Kermit>
```

At the `MS-Kermit>` prompt, you could type Kermit commands if you knew what they were. If an error message:

```
Bad command or file name
```

appears instead of the `MS-Kermit>` prompt, go back and read Chapter 2.

Some good commands to try are EXIT or QUIT, either of which will get you out of the MS-DOS Kermit program. When you type either command at the MS-Kermit> prompt, and then press the Enter key, you will return to the DOS prompt:

```
MS-Kermit>exit
or
MS-Kermit>quit
```

Practice going back and forth between DOS and Kermit a few times. Remember, the part you type is underlined, and you should terminate the underlined command by pressing the Enter key:

```
C>kermit
MS-Kermit>exit
C>kermit
MS-Kermit>quit
C>kermit
MS-Kermit>ex
C>kermit
MS-Kermit>q
C>
```

That wasn't too hard, was it? Notice that you can abbreviate EXIT as EX and QUIT as Q.

Menu on Demand

Unlike many other PC applications, Kermit does not present you with a full-screen menu, brightly colored windows, or sound effects when you run the program. It could, but it doesn't for reasons you may already understand if you make frequent use of menu-oriented software programs. Instead, the Kermit program gives you a prompt, MS-Kermit>, to indicate it is ready for a command. In response, you type a command. Kermit executes it and then gives you the next prompt. This goes on until you exit from the program.

Kermit's style is designed to be consistent across the hundreds of different kinds of computers and operating systems that Kermit programs run on. Once you have mastered one Kermit program, you have virtually mastered them all. And once you are familiar with the commands you need, you are not slowed down by screens full of menu options.

A typical Kermit command is an English verb, like HELP or EXIT. Some commands must be followed by additional words, known as *operands* or *parameters*: SHOW MODEM, CLOSE SESSION, and the like. Commands may even consist of several words: SET SPEED 2400, IF FAILURE GOTO JAIL. The command is terminated, and begins to execute, when you press the Enter key.

These commands resemble English sentences, and most of them make sense to the casual reader. As you might guess, there are hundreds of possible combinations of commands

and operands. Without a menu, and without thumbing through a thick reference manual, how do you know how to type a command?

The key that unlocks the mystery of Kermit's commands is the question mark. At the `MS-Kermit>` prompt, you can type a question mark at almost any point, and Kermit will tell you what is possible or expected next. If you type a question mark in response to Kermit's prompt, Kermit will list all its major commands. If you type a major command followed by a question mark, Kermit will tell you what sort of operand to type. For example, let's begin by seeing which SET command options begin with the letter S, and then let's pick one and find out what *its* options are, and so on, until we've completed a command:

```
MS-Kermit>set s? One of the following:
  send   server  speed
MS-Kermit>set server ?  One of the following:
  login  timeout
MS-Kermit>set server timeout ? Seconds
MS-Kermit>set server timeout 0
MS-Kermit>set server timeout 0 ? Press ENTER to execute command
```

In this way, you can feel your way through a command by typing a question mark within each field. Notice that previously typed characters do not need to be retyped.

Editing Your Commands

You will see a dashed line or rectangular block blinking at you when you are using DOS or software programs, like MS-DOS Kermit; this is called the *cursor*. As you type, the cursor will move along like the bouncing ball used on TV shows to help the viewers follow along. The cursor positions itself where you stopped typing so that you can always tell where you left off.

MS-DOS Kermit lets you edit your commands while you are typing them, as long as you haven't yet pressed the Enter key. Editing is done using the Backspace key or control characters. To produce a control character, hold down the key marked Ctrl and push the indicated key. For example, to type a Ctrl-C character, hold down the Ctrl key and press the letter C (uppercase or lowercase, it doesn't matter). Here are MS-DOS Kermit's command editing characters:

Backspace Deletes the *character* to the left of the cursor. May be typed repeatedly to delete characters all the way back to the `MS-Kermit>` prompt.

Ctrl-H Works the same as Backspace.

Ctrl-W Deletes the *word* to the left of the cursor. May be typed repeatedly to delete words all the way back to the `MS-Kermit>` prompt.

Ctrl-U Deletes the entire *line* all the way back to the `MS-Kermit>` prompt.

Ctrl-C Cancels the current command and returns immediately to the `MS-Kermit>` prompt.

Esc Completes the current word automatically if possible, and then goes on to the next word. If the rest of the current word cannot be guessed, the program emits a beep.

Summary of MS-DOS Kermit Command Features

- *Help is available for all fields.* You may type a question mark at any point within a command to get a short menu, except inside a filename (where a question mark acts as a wildcard character).
- *Kermit commands are case independent.* Commands and filenames may be in either uppercase or lowercase, or any mixture of them. Case does not matter.
- *Kermit commands may be abbreviated.* Any command word (but not filenames) may be abbreviated by omitting characters from the end as long as the abbreviation is different from any other word that can appear in the same field. For example, SER is sufficient to abbreviate SERVER, RU to abbreviate RUN, and SP to abbreviate SPEED.
- *Kermit command fields can complete themselves.* If you press the Esc key while typing a command word (but not a filename), and if the word is uniquely specified, Kermit will complete the word for you and position itself for the next word, if any. If the abbreviation is ambiguous, or if the word is a filename, Kermit will beep, and you may continue typing.
- *Kermit commands are not executed until you press the Enter key.* You always have the chance to ponder your command before activating it.
- *Kermit commands may be edited before entry.* While typing a command, you may use the Backspace key to erase characters back to the prompt. And you may press `Ctrl-W` to erase a word, `Ctrl-U` to erase an entire command, or `Ctrl-C` to cancel a command.

You should now take a few minutes to get comfortable with Kermit's command style, and the best way to do this is to get some practice. Following along on your PC, at the DOS prompt:

1. Type `kermit`. Did the Kermit program start? If not, did you remember to press the Enter key? If you did press the Enter key, you probably have not installed the program correctly. Go back to Chapter 2, and pay attention this time!
2. At the `MS-Kermit>` prompt, type `?` (question mark). You will see a list of all the MS-DOS Kermit commands, with a brief description of each. You will probably

never need most of these commands, but each has its purpose. (See Chapter 16 for a complete description of these commands.)

3. At the MS-Kermit> prompt, type set, followed by ? (question mark). These commands give you ways to change Kermit's behavior. Don't be frightened; usually Kermit behaves correctly without any extra instruction. Now press Ctrl-U to erase the SET command.

4. Type the command set terminal vt102 one letter at a time, following each letter with ? (question mark), and watch how the choices narrow down.

5. Now type the same command again, but this time press the Esc key after each letter instead of a question mark.

6. Type exit, and then press the Enter key to quit Kermit and return to the DOS prompt.

Kermit Startup Options

As you have seen, Kermit is normally an *interactive* program. You talk to it, and it talks to you. You can also make it act like a typical DOS "one-liner" command if you include Kermit commands on the DOS command line that invokes Kermit, for example:

```
C>kermit set speed 19200, send oofa.c
```

Multiple commands are separated by commas. When invoked in this manner, Kermit will execute all the commands on the command line and then exit to DOS without ever showing its prompt.

If you want to give command line options, but still get the prompt, include the STAY command:

```
C>kermit set speed 19200, send oofa.c, stay
```

You can use this feature to write DOS batch commands that start Kermit in special ways. For example, the file C.BAT might contain the line:

```
kermit set speed 9600, connect, stay
```

If C.BAT is stored somewhere in your PATH, just typing the C command at the DOS prompt will start Kermit and connect you to the host.

On the MS-DOS Kermit diskette, you will find a file called MSKERMIT.INI. This is the program's initialization file and includes Kermit commands. Each time you start up Kermit, these commands are executed automatically so that you don't have to type them at the MS-Kermit> prompt. This file must be stored in your PATH or in your current directory.

Some Basic Kermit Commands

You can experiment with some Kermit commands right away since they don't involve data communication at all. MS-DOS Kermit has some DOS commands built in. These include CD, DIR (which you can spell out as DIRECTORY if you like), TYPE, and DELETE:

```
MS-Kermit>dir oofa

 Directory of  C:\

OOFA     TXT      9753   9-21-89   8:43p
OOFA     OBJ       864  10-17-89   4:24p
OOFA     EXE      2861  10-17-89   4:24p
        2 File(s)     720 bytes free

MS-Kermit>type oofa.txt

   (The file OOFA.TXT is displayed)

MS-Kermit>del oofa.obj
MS-Kermit>cd \kermit
```

And the RUN command lets you run absolutely any DOS command or program from within Kermit, provided your PC has sufficient memory. Just type the desired DOS command after the RUN command, and then press the Enter key:

```
MS-Kermit>run basic
MS-Kermit>run rename oofa.txt secret.txt
MS-Kermit>run dir | find "<DIR>" | sort
```

Before proceeding to Chapter 7, practice using the features of Kermit's command processor. Get used to using a question mark, abbreviations, *Ctrl-U*, *Ctrl-W*, *Ctrl-C*, Esc, Backspace, and Enter so that the examples to come will be easy for you.

Chapter 7

Getting Online

We have seen a sample of many MS-DOS Kermit commands but have not explained which ones are absolutely needed and which are there to make life easier.

Like our ecclesiastical friends, the priest, the rabbi, and the minister, computers of different persuasions must adapt to one another's quirks. Fortunately, a PC equipped with Kermit is flexible and open-minded—ecumenical to a fault. Whenever it must communicate with a less tolerant computer of a different faith, MS-DOS Kermit will happily "convert" given the proper instruction.

The magic word is SET. SET this, SET that, and poof! Please pardon the lack of detailed explanation of the following tenets of data communications. These are mysteries you need not bother with to use MS-DOS Kermit.

Setting Communication Parameters

As you might expect, the Kermit command SET will define or change the characteristic you specify. The characteristic it defines or changes is called a *parameter*. There are several parameters you can set. You already know about the parameter called SPEED. In Chapter 5, you learned that the MS-DOS Kermit program and your modem must use the same speed in order to communicate and how to instruct MS-DOS Kermit to use the correct speed:

```
MS-Kermit>set speed 1200
```

Kermit's SET parameters have names like SPEED and BELL. Each of these parameters can have options too. The parameter options are referred to as *values*. In the example above, the parameter that is SET is SPEED, and the value it is set to is 1200.

Certain parameters are essential to establishing a successful connection if both terminal emulation and file transfer are to work correctly. Kermit parameters are set automatically, without any interaction from you, when the MS-DOS Kermit program is started. These startup values are called the *default* parameters. A Kermit SET command can override these default parameters either directly at the `MS-Kermit>` prompt or when included in your `MSKERMIT.INI` initialization file.

The most important parameter of all is the communication port. Unless you tell Kermit otherwise, it will use COM1, communication port number 1. COM1 can be either a serial port or an internal modem. If you need to use something else, because, for example, you have a serial port that Kermit doesn't recognize or you are connecting to a network, other options are available (see page 187). Let it never be said that Kermit limits your options.

Accessing the Port

An IBM PC or PS/2 may have more than one communication port. These ports are known by the DOS device names COM1, COM2, COM3, and COM4 and may be any combination of serial ports and internal modems.

Serial ports and internal modems made for IBM PCs, PC/XTs, and PC/ATs can be configured as either COM1 or COM2. This is usually done on the card itself by setting tiny switches or by plugging or unplugging a small jumper block. If you have only one communication port, you should make sure it is configured as COM1. If you have two, you should make sure that one of them is set up as COM1 and the other as COM2. Serial ports and internal modems for the PS/2 are self-configuring and, presumably, cannot be installed incorrectly.

For MS-DOS Kermit (or any other software) to communicate successfully, it must know which port to use. More often than not, COM1, which Kermit tries to use unless you say otherwise, is the correct choice. If you did not install the board yourself, the only way to know for sure if your serial port is COM1, COM2, COM3, or COM4 is by trial and error.

SET PORT COM1

(or SET PORT 1) Select communication port 1. You may also specify COM2, COM3, or COM4 (or 2, 3, or 4) if you are using a different port. If the port is an IBM serial port, or an internal modem that perfectly mimics a serial port, Kermit will control it directly for high-speed, efficient operation. Otherwise, Kermit will use the services of IBM's Basic Input Output System (BIOS), which results in slower operation. The

default port is COM1, meaning COM1 will be used if you don't issue a SET PORT command at all. See page 187 for other SET PORT options.

If your PC has more than one port, you must be sure that Kermit is trying to use the correct one. If Kermit is using COM1 but your cable is connected to COM2, there will be no communication. You must tell Kermit to use COM2 instead, for example:

```
MS-Kermit>set port com2
or
MS-Kermit>set port 2
```

The SET PORT command has a special property. The communication settings listed below—speed, duplex, flow control, handshake, and parity—are remembered for each port. The related SET commands affect the current port only. If you issue a SET PORT command to change ports, you get the new port and all its communication parameters. You can switch among ports as often as you like using only the SET PORT command.

Once you have chosen the communication port, you must be sure that the communication parameters that affect it are set appropriately:

SET SPEED

Two data devices cannot communicate *at all* unless they use the same transmission speed. The common speeds are 300, 1200, or 2400 bits per second (bps), or baud, for dialup (modem) connections; 4800, 9600, or 19200 bps for local PC-to-host connections; and 38400 or 57600 bps for short, direct PC-to-PC connections. Under exceptional conditions, two PCs might be able to communicate through a very short shielded cable at 115200 bps. MS-DOS Kermit's speeds can be abbreviated, just like words. For example, since 9600 is the only speed that begins with 9, SET SP 9 is equivalent to SET SPEED 9600.

```
MS-Kermit>set speed ? One of the following:
  45.5  50  75  110  134.5  150  300  600  1200  1800
  2000  2400  4800  9600  19200  38400  57600  11520
MS-Kermit>set speed 9
```

The transmission speed applies only to COM ports, not to BIOS or network connections. MS-DOS Kermit does not have a built-in default for transmission speed. It uses whatever speed the port was left at last time it was used unless it is told otherwise.

SET DUPLEX

Data connections can be either *full duplex* or *half duplex*. Full duplex means that both computers are allowed to transmit at the same time, and each computer is capable of simultaneously sending and receiving data, much like a conversation between two New Yorkers. This style of connection is the most common. It is used by UNIX systems, DEC VAX computers with VMS, and many others. On a full-duplex terminal

connection, the characters you type on your keyboard are sent to the other computer, and it "echoes" them back to you. This is called *remote echo.*

On a half-duplex connection, often used with IBM mainframes, only one computer is allowed to transmit at a time, much like CB radio. The two computers must take turns, and the characters you type are echoed to your screen by your own terminal (or in our case, by MS-DOS Kermit). This is called *local echo.*

The pertinent Kermit command is SET DUPLEX, with the operands FULL (for full duplex) and HALF (for half duplex). If you fail to SET DUPLEX HALF on a half-duplex connection, data sent out of turn will be lost.

```
MS-Kermit>set duplex ? One of the following:
  full  half
MS-Kermit>set duplex half
```

SET FLOW-CONTROL

Back in Chapter 3, did you try using `Ctrl-S` (hold down the Ctrl key while pressing the letter S) and `Ctrl-Q` to stop and resume the display of a long file during the DOS TYPE command? If not, try them now. Two computers with a full-duplex connection can be configured to do this to each other automatically. The process is called *Xon/Xoff flow control.* If computer A is receiving data faster than it can handle it, it sends an Xoff signal (Ctrl-S) to computer B, and computer B stops transmitting. When computer A has caught up, it sends an Xon (Ctrl-Q) to computer B, and computer B resumes transmitting. The Kermit command for this is SET FLOW. The operands are XON/XOFF and NONE. You should use Xon/Xoff flow control if the other computer supports it. To test this, connect to the other computer, TYPE a file, and then try typing `Ctrl-S` and `Ctrl-Q` manually to see if these characters suspend and continue the display.

```
MS-Kermit>set flow ? One of the following
  none, xon/xoff
MS-Kermit>set flow xon/xoff
```

SET HANDSHAKE

On a half-duplex connection, where only one computer is allowed to transmit at a time, there must be a signal by which each computer turns over transmit permission to the other. This is called a *line turnaround handshake.* The terminal or PC grants permission by sending a carriage return (for example, when you press the Enter key). The computer on the other end uses another special character (usually not a carriage return), such as Xon (Ctrl-Q). The Kermit command is SET HANDSHAKE. The operands are NONE, XON, and several others.

```
MS-Kermit>set handsh ? One of the following:
  xon, none ... (several others)
MS-Kermit>set handshake none
```

SET PARITY

Our final mysterious parameter is *parity*. Computer data is usually stored in chunks of eight bits. A bit is the basic unit of information in a computer and can have only two values: 0 and 1. An eight-bit chunk is called a character, or byte. The byte is the basic unit of transmission. But here's the tricky part. Although many computers transmit all eight bits of each byte, others transmit only seven and use the eighth bit for a kind of error check called parity. You must match MS-DOS Kermit's parity with that used by the other computer. The command is SET PARITY, and the operands are NONE, EVEN, ODD, MARK, and SPACE.

```
MS-Kermit>set parity ? One of the following:
  none, even, odd, mark, space
MS-Kermit>set parity none
```

Remember, the SET commands for speed, duplex, flow control, handshake, and parity apply to the currently selected port—the port you specified in your most recent SET PORT command, or else COM1 if you have not given a SET PORT command.

Before you can communicate successfully with another computer, you must find out its communication parameters and set them appropriately in MS-DOS Kermit. Table 7-1 should help. (A few entries are left blank in case you want to fill them in with your own favorite computer hosts.) Speed is not shown in the table because it depends on your connection method—1200 bps modem, 2400 bps modem, direct connection, and so on.

Sometimes you will have to set a lot of parameters yourself, and sometimes the default settings will apply. It depends on the computer you want to communicate with and the method of connection you choose. To set up MS-DOS Kermit for a direct 9600 bps connection to a DEC VAX/VMS system, the only parameter you must specifically SET is SPEED:

```
MS-Kermit>set speed 9600          (All defaults apply)
```

At the other extreme, to prepare for a 2400 bps dialup linemode connection to an IBM mainframe, there are many parameters you must SET:

```
MS-Kermit>set speed 2400          (Set the speed)
MS-Kermit>set duplex half         (Half duplex)
MS-Kermit>set flow none           (No flow control)
MS-Kermit>set handshake xon       (Xon handshake)
MS-Kermit>set parity mark         (Mark parity)
```

The communication settings used in Table 7-1 and the examples above are typical but may differ from those used at your site. Check the documentation for the computer you will actually be connecting to. This information should be readily available from the people who maintain that computer.

Table 7-1 Typical Communication Parameters

Environment	*Duplex*	*Flow*	*Handshake*	*Parity*
Default	Full	Xon/Xoff	None	None
UNIX	Full	Xon/Xoff	None	None
VAX/VMS	Full	Xon/Xoff	None	None
PDP-11	Full	Xon/Xoff	None	None
IBM Mainframe Fullscreen	Full	Xon/Xoff	None	Even
IBM Mainframe Linemode	Half	None	Xon	Mark
MS-Kermit to MS-Kermit	Half	Xon/Xoff	None	None
Direct Dial Services	Full	Xon/Xoff	None	None
X.25 Public Network	Full	Xon/Xoff	None	Mark
Local Area Network	Full	Xon/Xoff	None	None
?				
?				
?				

Dialup services like CompuServe and Dow Jones will generally use the parameters listed above under Direct Dial Services, but the parity setting might be something other than NONE. Check your subscriber literature. Since there are only five possible parity settings, you can try each until you find the one that works.

Note the entry for X.25 Public Network. These are services like Telenet and Tymnet in the United States, Datapac in Canada, and Datex-P in West Germany. These networks generally provide only seven-bit transmission and mask the other characteristics of the remote computer from you. Again, the parameter settings might be somewhat different, so check your subscriber literature.

When MS-DOS Kermit is operating through a local area network (LAN), whether it is DECnet, IBM Token Ring, or a vendor-supplied TCP/IP interface, the connection will almost always use the default parameters (see page 187).

What Happens If the Parameters Are Wrong?

You must adjust the communication parameters on your PC to match those of the computer or service you are connecting to. Although your parameters may seem correct during terminal emulation, only successful file transfer will tell for sure. There are some parameters that immediately identify themselves when you try to use terminal emulation.

COMMUNICATION PORT

If you are using the wrong communication port, you will not be able to communicate with your modem or the remote computer at all. You might see a message like `Serial port COM3 not available`. Or you may notice that you are not getting the proper responses after you type the Kermit CONNECT command, for example, if a Hayes modem does not respond OK after you type AT. Try a different port, for example, SET PORT 2 instead of SET PORT 1.

TRANSMISSION SPEED

If your transmission speed (baud rate) is set wrong, the speeds of the two computers will be mismatched, and you will see incomprehensible garbage on your screen, something like:

```
zj:Vo{{:n{::j{{:zbCzRrzrrz~{:~{{{{zqr::zj
```

Change your speed using SET SPEED. Try different speeds until you see comprehensible messages.

DUPLEX

If you have the wrong duplex setting, echoing will not be normal. If MS-DOS Kermit is set to full duplex (local echo off) when the connection is really half duplex, the characters you type will not appear on your screen at all, even though the responses to your commands will appear normally. Solution: SET DUPLEX HALF. If the connection is full duplex and MS-DOS Kermit is set to half, every character you type will appear twice, for example:

```
wwhhaatt''ss ggooiinngg oonn hheerree??
```

Solution: SET DUPLEX FULL.

FLOW CONTROL

On a full-duplex connection, losing data can be prevented by using Xon/Xoff flow control, but only if both computers are set up to do it. MS-DOS Kermit does this by default, and so do most computers, including UNIX-based computers, VAX/VMS, and others. If MS-DOS Kermit is doing Xon/Xoff but the other computer is not, an Xoff signal (Ctrl-S) sent by Kermit to the other computer will be misinterpreted, with unpredictable consequences. Solution: Tell MS-DOS Kermit to SET FLOW NONE.

If MS-DOS Kermit is not doing flow control, there is a danger of loss of data during long, scrolling screen displays at high speeds and when Kermit is printing screens or saving them to a file. Flow-control signals sent by the other computer under these conditions will be accepted by MS-DOS Kermit as ordinary data characters. Solution: Tell MS-DOS Kermit to SET FLOW XON/XOFF.

PARITY

MS-DOS Kermit tries to ignore parity as much as possible. During terminal emulation, MS-DOS Kermit strips the parity bit from each character so that even when its parity is set to NONE and the other computer is sending, say, MARK parity, the display will appear correct. The problem is usually in the other direction. If the computer on the other end, or some piece of equipment between the two computers, *needs* a certain kind of parity, and if MS-DOS Kermit doesn't send it, certain characters will not arrive correctly at their destination. So, we cannot ignore parity completely during terminal emulation, and we must pay attention to it during file transfer.

Chapter 8

Terminal Emulation

The world is bursting with computers and data services that possess a wealth of information and applications just for you. Most of these computers are designed to be accessed by terminals. But you have a PC. Luckily for you, your PC—when equipped with MS-DOS Kermit—can do everything a terminal can do, and much more.

Kermit's job is to create the best possible connection between your PC and the host computer. But because there are so many different kinds of computers, so many different kinds of terminals, and so many different ways in which terminals and computers communicate, this job is not always simple.

The Mechanics of Terminal Emulation

Once you give the CONNECT command at the `MS-Kermit>` prompt, you have begun terminal emulation; your PC has become a terminal to the other computer. Terminal emulation gives you the impression that you are typing directly to the other, remote computer instead of to the PC on your desk.

Before you can use Kermit as a terminal, you must set all of your communication parameters appropriately, as explained in Chapter 7. Once this is done, you can issue Kermit's CONNECT command. CONNECT is such a frequently used command that it has a special one-letter abbreviation, C. Once connected, any commands you type will be ignored by your PC and interpreted by the remote computer. For example, if you type the

command DIR at the MS-Kermit> prompt, you will see a list of the files on the PC (in your current directory). Remember that none of the commands you type will be recognized until you press the Enter key.

```
C>kermit                          (Start up Kermit on your PC)
MS-Kermit>cd \kermit              (Change directory to \KERMIT)
MS-Kermit>dir                     (Request a list of files)

 Volume in drive C is EASYDISK
 Directory of  C:\KERMIT

KERMIT   EXE    112416   9-10-89    9:47p
MSKERMIT INI      3265   8-08-89    8:31a
KERMIT   PIF       165  12-09-89    5:36p
        3 File(s)    1472512 bytes free
```

After you have typed the CONNECT command at the MS-Kermit> prompt and have given the proper codes to gain access to the remote computer, a DIR command typed there will give completely different results.

Gaining access to a computer with more than one user is called *logging in*, and ending your access to this computer is called *logging out*. The access codes generally consist of a *username* and a *password*, which are issued to you by those who manage the multiuser computer.

```
C>kermit                          (Start up Kermit on your PC)
MS-Kermit>set sp 9600             (Set the right speed)
MS-Kermit>set parity even         (Maybe other parameters too)
MS-Kermit>connect
Welcome to the Other Computer, VMS V5.0-2
Username: kim                     (Type your user ID)
Password: ________                (Type your password)

$ set term/inquire                (Set your terminal type)
$ dir                             (Type the DIR command)

Directory $DISK1:[KIM]

CALIF.DIR;1  CKERMIT.INI;9 EDTINI.EDT;6    EMACSINI.EL;1
LOGIN.COM;6  MAIL.MAI;1    MOON.DOC;5      OOFA.C;44
OOFA.OBJ;35  OOFA.EXE;35   VMSKERMIT.INI;8

$ logout                          (Remember to log out!)
```

These are entirely different files on a completely different computer.

When you CONNECT to the other computer, you are actually using your PC keyboard to talk to *two* computers—your PC and a remote computer. You need some way to shift the

conversation from one computer to the other. The standard method is to type a special *escape sequence*, `Ctrl-]C`. That is, hold down the Ctrl key, press] (the right bracket key), let go of the Ctrl and] keys, and then press the letter C (for Close Connection). Practice this a few times:

```
MS-Kermit>set speed 9600          (Set the right speed)
MS-Kermit>connect                 (Begin terminal emulation)

$ Now I'm communicating with the host computer.
Command not found: "Now"
Ctrl-]c                           (Escape back to Kermit on the PC)
MS-Kermit>connect                 (Connect to the host again)
$ help
UNIX helps those who help themselves.
$ thanks a lot!
Command not found: "Thanks"
Ctrl-]c                           (Escape back to Kermit)
MS-Kermit>
```

When you reconnect, your previous terminal screen reappears. Escaping back (returning) to the PC will not terminate your session on the other computer. You must use whatever commands are necessary to end your session on the other computer. (A typical command for this purpose is LOGOUT.) If you are paying per minute for the connection, this is an important fact to remember. Even if time on the remote computer doesn't cost you a cent, keep in mind that turning off your PC for the night does not necessarily mean the connection to the other computer is gone. It is indeed possible for anyone to start up your PC, run a copy of the MS-DOS Kermit program, type CONNECT, and access your host session. This is not a good thought if your manager is that "anyone" and your resume is the file you were working on!

The Mode Line

Also notice the all-important *mode line* at the bottom of your screen, which looks approximately like this:

```
Esc-chr:^] help:^]? port:1 speed:9600 parity:none echo:rem VT320
```

This gives you the vital statistics on your terminal session. It tells you that your escape character is `Ctrl-]`,[9] you can get help about connect-mode escape sequences by typing

[9]Control characters are often written as ^ followed by a letter or symbol, for example, `^A` for Ctrl-A. They are often displayed in this way by host computers and even by DOS. To see this, type `Ctrl-A` at the DOS prompt.

Ctrl-] followed by a question mark, your communication port is COM1, your transmission speed is 9600 bps, parity is none, character echo is being done remotely (full duplex), and Kermit is emulating a VT320 terminal.

Connect-Mode Escape Options

The escape character, *Ctrl-]*, can take other operands besides C. You can get a list of them by typing *Ctrl-]?*, just as it told you in the mode line above. For example, *Ctrl-]P* will "push" to DOS. You will get a DOS prompt, and you can carry on with DOS as long as you like without forgetting your parameter settings for the host or terminating the connection. When you want to return to Kermit, type the DOS command EXIT (which is the same as the MS-DOS Kermit command), and you will be back in your terminal session, exactly where you left off, with your previous screen intact.

Another useful connect-mode command is *Ctrl-]F*, which will copy ("dump") the current screen into a file on your DOS disk. The name of this file on your DOS disk is KERMIT.SCN, but you can select a different file with Kermit's SET DUMP command. To do this, return to the MS-Kermit> prompt by typing *Ctrl-]C* and give the command SET DUMP and a filename. Then issue the CONNECT command again, and type *Ctrl-]F:*

```
$
The screen looks like this on the other computer.
It is filled with lines of text.
And commands.
$
Ctrl-]c                          (Escape back to Kermit on the PC)

MS-Kermit>set dump amm.scn       (Select a new screen copy file)
MS-Kermit>connect                (Connect to other computer again)

Ctrl-]f                          (Ctrl-right bracket followed by F)
                                 (to copy screen to PC file AMM.SCN)
```

If the screen-copy file already exists, new material is added to the end, with each screenful separated by a Ctrl-L (formfeed) character. Otherwise, a new file is created the first time you copy the screen. To find out how you can save more than one screenful of data, see the section "Screen Rollback" later in this chapter.

Table 8-1 lists all the MS-DOS Kermit connect-mode escapes. The BREAK signal, which Kermit sends if you type *Ctrl-]B* (hold down the Ctrl key and press right bracket, then press the letter B) is not a normal character but a special signal lasting about one-quarter second that is required by some hosts or communication processors. A long BREAK is the same thing, but it is longer.

Table 8-1 MS-DOS Kermit Connect-Mode Escapes

Escape	*Function*
`Ctrl-]?`	Display a list of escape functions
`Ctrl-]0`	(the digit zero) Transmit a NUL character
`Ctrl-]B`	Transmit a BREAK signal
`Ctrl-]L`	Transmit a long BREAK
`Ctrl-]C`	Return to the MS-Kermit prompt
`Ctrl-]H`	Hang up the phone; terminate a modem connection
`Ctrl-]F`	Copy the current screen into a DOS file
`Ctrl-]M`	Turn the mode line off if it is on; turn it on if it is off
`Ctrl-]P`	Push to DOS (use DOS EXIT to return to Kermit's terminal screen)
`Ctrl-]Q`	Temporarily quit logging the terminal session
`Ctrl-]R`	Resume logging the terminal session
`Ctrl-]S`	Show the status of the connection
`Ctrl-]Ctrl-]`	(two copies of the escape character) Send the escape character itself

Kermit's connect-mode escapes are hard to type. This is done on purpose to prevent you from accidentally invoking a function like hangup. For convenience, some of them also have Alt-key equivalents, and certain other functions are also assigned to Alt keys. Table 8-2 shows some of the Alt-key functions. To invoke an Alt-key function, hold down the Alt key and press the indicated key. For example, to return from terminal emulation to the `MS-Kermit>` prompt, hold down Alt and press X.

Table 8-2 MS-DOS Kermit Alt-Key Commands

Alt Key	*Function*
`Alt-X`	Exit from connect mode back to the MS-Kermit prompt
`Alt--`	(Alt-minus) Change Kermit's terminal type
`Alt-=`	Reset the terminal and clear the screen
`Alt-B`	Transmit a BREAK signal

Do I Need a Terminal Emulator?

Kermit's well-deserved reputation as a reliable and efficient file transfer protocol may mislead you into thinking that you need another package for high-quality terminal emulation. You don't.

If your only interest is transferring files between your PC and the remote computer, you probably know all you need to know about terminal emulation—just enough to get you connected. If so, go on to Chapter 9 and read about file transfer. But if you are connecting to a bulletin board or dialup service, or if you plan to actually use the remote computer to do work, you should become familiar with Kermit's terminal emulation features.

Terminal Emulation Options

Different terminals have different characteristics. The appearance of their screens is controlled in different ways by the computers they communicate with. For your screen to be formatted correctly, Kermit must emulate the kind of terminal that the remote computer believes it is controlling. MS-DOS Kermit can emulate any of the following terminals:

DEC VT102
Similar to the industry standard DEC VT100 but with added editing capabilities. Features include direct cursor positioning, ability to partition the screen into separate scrolling regions, character highlighting (boldface, underscore, inverse video), line and screen erasure, line and character insertion and deletion.

DEC VT320
A more advanced DEC terminal that does everything the VT102 does but also includes the ability to switch among different character sets, to enter international characters from the keyboard, and to redefine keys under host control. In addition, the DEC function keys are supported. The VT320 is also compatible with the DEC VT200 and VT100 series. Unless you say otherwise, MS-DOS Kermit emulates the VT320 terminal.

DEC VT52
An old DEC terminal with only the bare essentials.

Heath/Zenith-19
A combination of VT52 and VT102 features.

Tektronix 4010
A graphics terminal capable of drawing pictures on the screen.

Most computers, such as UNIX and VAX/VMS systems, support a variety of terminals. Certain applications on these computers expect to be able to control the appearance of

your screen. Examples include EMACS or VI on UNIX, PHONE or EDT on VMS, and IBM mainframe protocol converters in general. The same is true for many dialup services, like the Digital Electronic Store that we visited earlier. For all this to work, Kermit's terminal type must agree with the host's.

The Kermit command is SET TERMINAL *type*, where *type* can be HEATH-19, VT52, VT102, VT320, or TEK. There is also a special type, NONE, that tells Kermit to ignore all screen formatting commands from the host. The Kermit command is entered like this:

```
MS-Kermit>set terminal vt102
```

Which Terminal Type Should I Use?

If your host supports the VT320, use that because it has the most advanced features; then, in order of preference, the VT102, Heath/Zenith-19, and VT52. If your host supports VT100 but not VT102 or VT320, use VT102. If your host supports VT200 but not VT320, use VT320. If you want to do graphics, use Tektronix. If your host supports none of these, use NONE. Among the more advanced features of the VT102 and VT320 is the ability of full-screen applications to update the screen more efficiently, which is desirable on a low-speed dialup connection.

How Do I Tell the Host Which Kind of Terminal I Have?

The answer to this question is different for every kind of host. Let's look at some common cases.

VAX/VMS

First set Kermit to the desired terminal type—VT320 for release 5.0 and later of VMS, and VT102 for older releases. Then, after CONNECTing and logging in to VMS, type the VMS command SET TERMINAL/INQUIRE.[10] VMS will ask your terminal what it is, and it will respond automatically. Then type the VMS command SHOW TERMINAL to make sure the inquiry was answered properly. If not, try again. Here's an example:

```
Username: matt                    (Type your username)
Password: ______                  (Type password; it doesn't echo)

Welcome to VAX/VMS V5.0
```

[10]When using release 4.x of VMS, you can set Kermit's terminal type to VT320 and give the VMS command SET TERM/DEVICE=VT200.

```
$ set terminal/inquire
$ show terminal
Terminal: _TXA0:  Device_Type: VT300_Series  Owner: MATT
  Input:  9600  LFfill:  0  Width:  80  Parity: None
  Output: 9600  CRfill:  0  Page:   24
```

UNIX

For UNIX, the method depends on which kind of UNIX system you have, which shell you are using, and how things are set up at your site. In general, the trick is to set your TERM environment variable to the desired terminal type. In some cases, two commands are necessary (see Table 8-3). Note that case matters in UNIX commands: `VT102` is not the same as `vt102`.

You can also check that your command worked by giving the UNIX "clear" command, which should clear your screen. Some UNIX systems also support a command called "tset," which sets your terminal type as well as many other terminal characteristics:

```
eval 'tset -sQ vt102'
```

Type the UNIX command `man tset` for details.

IBM Mainframes

When connecting to IBM mainframes through protocol converters such as the Series/1 or 7171, you will be prompted for your terminal type. The names vary from site to site, so check your local documentation if you have difficulties:

```
ENTER TERMINAL TYPE: vt-100
```

You can usually check what terminal types are available by pressing the Enter key in response to the prompt:

Table 8-3 Setting Your Terminal Type in UNIX

Shell	*Command 1*	*Command 2*	*Check*
Bourne shell (sh)	`TERM=vt102`	`export TERM`	`echo $TERM`
C-shell (csh)	`setenv TERM vt102`	(none)	`echo $TERM`
Korn shell (ksh)	`TERM=vt102`	`export TERM`	`echo $TERM`

```
ENTER TERMINAL TYPE:              (Press the Enter key)
VALID TYPES ARE:
ADM-3A     H19
HP2621     IBM-3101
VT-100     VT-52
ENTER TERMINAL TYPE: vt-100
```

In this case, VT100 is the best choice because it has more features than than the VT52 or Heath/Zenith-19, and MS-DOS Kermit does not emulate any of the others. Terminal types on the IBM mainframe must be typed in exactly; notice the - (dash) between the `vt` and the `100`.

For IBM mainframe linemode connections, in which the host does not make any attempt to control the format of your screen, it doesn't matter which terminal type is used.

Terminal Characteristics

Besides the terminal type, there are many other terminal characteristics that you can control using Kermit's SET TERMINAL command. Whether you choose to use all, some, or none of the commands, you should be aware of your options. And there are so many options in the MS-DOS Kermit program! Here are some examples:

SET TERMINAL WRAP

The options are ON and OFF. This controls what Kermit does if it receives a line of text wider than the screen. When WRAP is ON, Kermit will break, or wrap, the line onto the next line. Otherwise, characters past the right edge of the screen will be lost. WRAP is OFF by default on the assumption that the host computer will do the line-wrapping itself. Example:

```
MS-Kermit>set term wrap on
```

SET TERMINAL TABSTOPS AT

Kermit's tabs come preset to every eight spaces, just like the DEC terminals that Kermit emulates. This command lets you change them. The operand is a list of numbers specifying where to put the tab stops, for example:

```
MS-Kermit>set terminal tabs at 10 20 30 40 50 60 70
```

You can also SET TERMINAL TABS CLEAR AT the specified numbers or SET TERMINAL TABS CLEAR ALL to remove all tab stops. If tabs are to be set at regular intervals, you can use a special notation: SET TERM TABS AT 1:10, which means set tabs every ten spaces, starting at screen column 1.

SET TERMINAL NEWLINE

Normally when you press the Enter key, Kermit sends a carriage return only. SET TERMINAL NEWLINE ON makes it send both carriage return and linefeed, which is useful for PC-to-PC communication.

SET TERMINAL DIRECTION

The options are LEFT-TO-RIGHT (the default) and RIGHT-TO-LEFT. The purpose of this command will be explained later. For now, it's just for fun. Try it!

There are also many other terminal settings. At the `MS-Kermit>` prompt, you can type:

```
MS-Kermit>set terminal ?
```

to find out what they are. Some will be explained as we proceed. The rest of the commands are listed in Chapter 16.

Screen Rollback

If you use terminal emulation at all, you will appreciate this feature. How many times have you wanted to look again at something that has already left your screen? MS-DOS Kermit gives you this ability. Just press the PC's PgUp (Page Up) key to see the previous screen. Press it several times to see several previous screens. Kermit's normal retention is about ten screens, but you can change its capacity by adding a Kermit environment variable to your `AUTOEXEC.BAT` file, for example:

```
SET KERMIT=ROLLBACK 20
```

to allocate 20 screens worth of rollback memory. (Each screen requires about 4K of memory.) After rolling back, you can roll forward again by pressing the PgDn (Page Down) key.

You can even roll back and forward a line at a time, rather than a screen at a time, by holding down the Ctrl key while you press PgUp and PgDn. And you can restore the latest (bottom, newest) screen instantly by pushing the End key. To go directly to the earliest (top, oldest) screen in Kermit's memory, press the Home key.

What should Kermit do if new characters arrive at the serial port while your screen is rolled back to some time in the past? It's your decision:

SET TERMINAL ROLLBACK ON

Restore the latest screen, and then display the new characters in their rightful place.

SET TERMINAL ROLLBACK OFF

Display new characters on the current position in the rolled-back screen. This is particularly useful if you rolled back your screen to copy something from an earlier screen.

Kermit's rollback feature is not only a convenience, it can be a lifesaver. Suppose you have been typing text into a host application—for example, a text editor or electronic mail program—for hours, when suddenly your connection to the host computer dies. Normally, all of your work would be lost. But Kermit can save you! Just press the Home key to get to the top of your screen memory, then type *Ctrl-]F* to copy the screen to a file, press PgDn, copy the next screen, and so on to the bottom of the screen memory. All of your work is now recorded in your screen-copy file, which you can transfer back to the host (just as soon as you learn how to use Kermit to transfer files).

Session Logging

Where there is a Kermit command, there is a reason. You might someday need to record your entire session, or selected parts of it, to a PC file. You can do this by giving the LOG SESSION command. Every character that arrives at the serial port (after you give MS-DOS Kermit the CONNECT command) will be recorded in the file SESSION.LOG. This method saves you the trouble of rolling back your screen and typing *Ctrl-]F* several times if you know ahead of time that you want this information.

You can also specify a different filename for the session log file:

```
MS-Kermit>log session who.log    (Select session logging)
MS-Kermit>connect                (Begin terminal emulation)
                                 (Press the Enter key)
$                                (The other computer's prompt)
$ who am i                       (Commands are copied to WHO.LOG)
watsun!karen                     (All responses go there too)
Ctrl-]c                          (Escape back to the PC)
MS-Kermit>close session          (Stop copying screen to file)
MS-Kermit>connect                (Go back to the other computer)
```

When you are finished logging, you can close the DOS file using the Kermit command CLOSE SESSION, or you can EXIT from the MS-DOS Kermit program. Session logging can be turned off without returning to the MS-Kermit> prompt by typing the connect-escape sequence *Ctrl-]Q* (hold down the Ctrl key and press the letter Q), and it can be resumed with *Ctrl-]R*. New material is written to the end of the log. Session logging differs from screen copying with *Ctrl-]F* in two ways:

1. Screen copy saves only the current screen, whereas session logging can save an entire session or any part of it.
2. Screen copy saves only the characters that appear on the screen, but session logging saves all characters sent by the host, including screen formatting commands.

A session log that has been saved to a DOS disk can be replayed. Assuming the name of the session log is SESSION.LOG and that your terminal type was VT102 during the session, the procedure is:

```
MS-Kermit>set term vt102
MS-Kermit>replay session.log
```

You will see the entire recorded session unroll before your eyes. This can be mildly entertaining for sessions that control the screen in fancy ways, such as the Digital Electronic Store tour or a Tektronix graphics session. This sort of display is not only for amusement, but can be useful too, for example, in demonstrating or marketing computer software.

The Kermit distribution diskette contains some files that can be displayed in this manner. Files of type .TEK should be played through the Tektronix emulator, those of type .VT through the VT-320 emulator, and those of type .HGR through the Heath/Zenith-19 emulator.

```
MS-Kermit>set term h19
MS-Kermit>replay castle.hgr
```

Key Redefinition

Have you ever noticed that every time IBM releases a new model PC, it has changed the keyboard so much that you can't find half the keys? Do you have to retrain your fingers several times a day as you switch between your PC and some other kind of keyboard? Are there certain words or phrases that you type so frequently that you wish they could be entered magically with a single keystroke?

If you answered yes to any of these questions, MS-DOS Kermit can help. Kermit lets you assign anything you like to any key at all. These definitions are active only during terminal emulation (after you have given Kermit the CONNECT command); they do not affect the commands you type at the MS-Kermit> prompt or DOS commands.

The Kermit command to establish a key definition is SET KEY. The easiest way to use this command is to type SET KEY, and then press the Enter key. Kermit will ask you to press a key, and then it will ask you to type the new definition for that key:

```
MS-Kermit>set key
  Push key to be defined:        (You press the F1 key)
  Enter new definition: Fooey!
  Scan code \315 is defined as
  String: Fooey!
```

From now on, every time you press the F1 key, Kermit will send Fooey! to the host computer, just as if you had typed it.

You can examine a key's definition by typing the SHOW KEY command at the MS-Kermit> prompt and then pressing the desired key when MS-DOS Kermit asks you to. Practically any key or key combination can be defined: a single key, a shifted key, a Ctrl-key combination, an Alt-key combination, a Ctrl-Alt-key combination, a Ctrl-Alt-Shift combination, and so on.

Key definitions can include control characters, but only if you specify them in a particular way. You are not allowed to type them literally because they have special meanings of their own to Kermit's command processor (for example, Enter terminates a command, and *Ctrl-C* interrupts a command). This special notation is the backslash character (\) followed by a number that tells what character it is. These numbers are listed in the ASCII[11] table (Table II-4) in Appendix II. For example, ASCII character 13 is CR (Carriage Return), which is produced on the PC by pressing the Enter key. So, if you wanted to have a fast way of logging out from the host, you could put a logout command, including the carriage return, on a single key, say F2:

```
MS-Kermit>set key
  Push key to be defined:        (You push the F2 key)
  Enter new definition: logout\13
  Scan code \316 is defined as
  String: logout^M
```

The ^M tells you that Kermit has correctly interpreted your \13 as carriage return, which is indeed Ctrl-M (see Table II-4).

Notice the phrase "scan code" that appears when you type a SET KEY or SHOW KEY command. Every PC key combination has a code that identifies it. The letter A is 65, *Ctrl-A* is 1 (one), *Alt-A* is 2334, *Ctrl-Alt-A* is 3358, and so on. If you know the scan code of the key you want to define, you can write the key definition command all on one line so that further interaction is not necessary:

```
SET KEY \317 help\13
```

This assigns the command HELP to the F3 key. To find out a key's scan code, type show key, and then press the desired key or key combination. For example, if you wanted to see what the scan code was for the letter a:

```
MS-Kermit>show key
Push key to be shown (? shows all): a   (You press the a key)

ASCII char: a   \97 decimal is defined as
Self, no translation.
```

[11]ASCII is the American Standard Code for Information Interchange, the code used by the PC and most other computers for representing characters.

The example shows that the scan code for the letter a is \97.

Perhaps the most popular of all SET KEY commands is the one that moves the Esc key from wherever IBM is putting it this year (far right of the keypad on the PC/AT keyboard, extreme upper left on the PS/2 keyboard) to the location that VT100 users are accustomed to—just left of the digit 1 and above the Tab key. The key that occupies this location is usually the ` (accent grave) key. If you type the SHOW KEY command and then push the ` key, you will learn that its scan code is 96. And if you look up ESC in Table II-4, you'll see that its decimal value is 27. So the following command will do the trick:

```
MS-Kermit>set key \96 \27
```

But now you no longer have a way to transmit an accent grave from the keyboard. The logical answer is to assign it to the Esc key:

```
MS-Kermit>set key \324 \96
```

MS-DOS Kermit's keyboard scan codes are listed in Table II-6 in Appendix II.

But that's not all! Not only can you assign any character to any key, and not only can you assign a string of characters to any key, but you can also assign Kermit *functions* to the keys of your choice. But before you can do this, you must know the names of the Kermit functions. Table II-3 in Appendix II lists them; it also shows the default key assignments for these functions. Here's one example:

```
MS-Kermit>set key \319 \Kexit
```

This assigns the escape-back-from-connect-mode function, which is called \Kexit and is normally assigned to *Ctrl-]C* or *Alt-X*, to the F5 key.

So that you don't have to type numerous SET KEY commands every time you start Kermit, you can put all your key definitions into your MSKERMIT.INI file, so they will take effect automatically each time you start Kermit.

Screen Color

If you have a color monitor, you can use Kermit's SET TERMINAL COLOR command to establish the foreground and background colors that will be used during terminal emulation. If you will be doing a lot of terminal emulation, this feature can help you find the color combination most soothing to your eyes. Moreover, if the color you choose is different from the color on your PC when you are not doing terminal emulation, it is an easy way to remember which computer you are conversing with.

The operands of the SET TERMINAL COLOR command include two-digit numbers that specify the colors. A number starting with 3 gives the foreground color, and a number starting with 4 gives the background color. The second digit tells the actual color to be used (see Table 8-4), for example:

Table 8-4 Kermit Screen Colors

Color	*Foreground*	*Background*
black	30	40
red	31	41
green	32	42
orange	33	43
blue	34	44
amethyst	35	45
turquoise	36	46
white	37	47

```
SET TERMINAL COLOR 34 47          (Blue on white)
```

Some of the color combinations are pretty wild—try them! If you include the number 1, you will get a high-intensity foreground, which also changes the color somewhat. Try this:

```
SET TERMINAL COLOR 1 31 45        (Purple and pink!)
```

To reset the foreground to normal intensity, include the number 0:

```
SET TERMINAL COLOR 0 31 45        (Hard to read!)
```

There are 128 possible combinations, but some of them make little sense (like black on black). If you specify the number 0 by itself, the result is white on black.

The SET TERMINAL SCREEN-BACKGROUND command lets you switch your screen's foreground and background colors with each other. The options are NORMAL and REVERSE. Remember that your colors can be seen only *after* you type the CONNECT command at the `MS-Kermit>` prompt. You will need other software for your PC to control the colors when you are not using Kermit for terminal emulation.

Printer Control

If you have a printer attached to your PC, you can use Kermit to control it in several different ways during terminal emulation. Pressing the PrtSc (Print Screen) key (together with the Shift key on some systems) will cause the current contents of the screen to be printed by DOS; holding down the Ctrl key while pressing the Print Screen key will alternately start and stop the copying of incoming characters to the printer.

Host-controlled printing is also supported; in this case, it is up to the host to send special printer control commands that will direct incoming characters to the printer rather than to the screen. The host can also direct the characters to both the printer and the screen. When the printer is activated during terminal emulation, the mode line will contain the letters PRN (the DOS device name for the printer) at the far right. If the printer is not ready (for example, not turned on), a message telling you so will appear in Kermit's mode line. If this happens, do whatever you normally do to get your printer ready, or type *Alt-=* (hold down the Alt key and press the equal-sign key) to reset the terminal emulator.

You can also direct the session log to the printer by specifying the device name of your printer in the LOG SESSION command, for example:

```
MS-Kermit>log session prn
```

You can temporarily stop printing by using *Ctrl-]Q* and resume it using *Ctrl-]R*, and you can close the printer's session log with the CLOSE SESSION command.

Finally, you can set your screen copy file to be the printer:

```
Kermit-MS>set dump prn
```

Any time you want to print your current screen, just use *Ctrl-]F*. With this option, you will get a formfeed (page eject) after each screen.

To prevent loss of characters while printing, your printer should have a parallel (rather than serial) interface, and Kermit should have a full-duplex host connection with Xon/Xoff flow control (see Table 7-1).

Graphics

If you are one of those people who believes that a picture paints a thousand words, then you'll like this feature. MS-DOS Kermit is capable of emulating a Tektronix 4010/4014 graphics terminal for use with host-based software that can generate Tektronix graphics images. Examples of this graphic capability are on the distribution diskette and have the filetype .TEK. You cannot create these graphic images with MS-DOS Kermit alone. The software on the other computer produces the graphic image—that's the graphics software. Kermit simply (actually not so simply) displays these images as the host constructs them or saves them in a file to be REPLAYed at a later time.

The Tektronix emulator implements a mixture of Tek 4010, 4014, and other features to draw characters, lines, dots, and rectangle-fill in graphics mode compatible with the DEC VT340 terminal's Tektronix mode. Most popular PC graphics adapters are supported, including CGA, EGA, VGA, AT&T/Olivetti, and Hercules. Kermit's connect-mode foreground and background colors will be used during graphics mode, and host-controlled

color graphics are also possible. Pure monochrome systems, of course, lack both graphics and color capability; in this case, Kermit approximates the graphic image by writing dots as monochrome plus signs.

Kermit tries to figure out what kind of graphics adapter you have automatically. In some cases, it might guess wrong. If it does, you can use the command SET TERMINAL GRAPHICS to tell it exactly which kind of adapter it will be using. Type the Kermit command `set terminal graphics ?` to find out which adapters are supported.[12]

When you enter Tektronix emulation, your cursor will disappear. Don't be alarmed; this is how real Tektronix terminals behave. To enter graphics mode, give the command SET TERMINAL TEKTRONIX, or let your host application put Kermit into graphics mode automatically if it can. If you escape back to the `MS-Kermit>` prompt and then reconnect, your graphics screen image will be restored only if your graphics adapter has sufficient memory to store it:

```
MS-Kermit>set terminal graphics EGA
MS-Kermit>set terminal tektronix
MS-Kermit>connect

THIS SCREEN INTENTIONALLY LEFT BLANK
```

Kermit can give you the illusion of having two windows—one with text and one with graphics—but with the ability to show only one window on your screen at a time. Once in graphics mode, you can get your text-mode screen back by typing Kermit's toggle terminal type key(s), normally *Alt--* (hold down the Alt key and press the minus key). Its contents are preserved for you during graphics operation. You can jump back to your graphics screen by typing *Alt--* a few more times. And you can clear your graphics or your text screen by using *Alt-=* (*Alt-Equals*) when either "window" is displayed on your screen.

If you hear a beep during Tektronix emulation, that means the host has sent picture elements that are outside the current screen boundaries. Look at the current picture until you're tired of it, and then press the Enter key to see the new material. The old picture will disappear from your screen, and the new picture will be displayed.

During your graphics session, a cross-hair cursor that looks like a large + (plus sign) might appear. This means that the host application wants you to move the cross hairs to

[12]Even then if the display adapter declines to reveal its true colors (pun intended), Kermit drops back to the simplest kind that might work.

some point on the picture and then press the Enter key. This will send the screen coordinates of the cursor to the host application. Move the cross-hair cursor using your keypad arrow keys, shifted for coarse movements and unshifted for fine tuning. If your PC has a mouse, you can use it to move the cross-hair cursor if you have a mouse driver installed in your system (for example, using the MOUSE ON and DEFAULT commands that come with the Microsoft Mouse). Push the right mouse button to send the cross hair's position to the host.

Kermit's screen copy feature (*Ctrl-]F*) does not work with graphics screens, but the graphics commands that the host transmitted to create the image can be recorded on your DOS disk by using Kermit's LOG SESSION command, and the resulting graphic images can be recreated by using Kermit's REPLAY command on the session log file:

```
MS-Kermit>log sess tek.log      (Record session in file TEK.LOG)
MS-Kermit>set term tek          (Set terminal type to Tektronix)
MS-Kermit>connect               (Begin terminal emulation)

(Create graphics screen)

Alt-X                           (Hold down the Alt key and press X)
MS-Kermit>close sess            (Close the session log)
MS-Kermit>replay tek.log        (View the graphics screen again)
```

Similarly, the connect-mode printer control keys do not normally work with graphics. To print graphics screens, you need a special print driver program, such as the GRAPHICS.COM file supplied by IBM with DOS, which works with CGA systems and IBM printers. A public-domain driver for EGA systems with Epson printers, EPSON.COM, is supplied on the Kermit distribution diskette. On black-and-white printers, colored screens may be printed as all black ink because all screen dots that are illuminated are printed. To get a sensible printout, set your background color to black (use the command SET TERMINAL COLOR 0). The foreground color doesn't matter.

Several files are included on the Kermit distribution diskette to demonstrate Kermit's Tektronix emulation. To use them, start the MS-DOS Kermit program, and then issue a REPLAY command for the file; for example:

```
MS-Kermit>set terminal tek
MS-Kermit>replay usa.tek
```

shows a map of the United States.

Despite its length, this chapter has only scratched the surface of Kermit's terminal emulator. A few additional features will be presented later on, but for a detailed treatment, consult the forthcoming book, *The Making of MS-DOS Kermit* by Joe R. Doupnik.

Chapter 9

File Transfer

The time has finally come to begin transferring files with Kermit. This is probably what attracted you to the MS-DOS Kermit program in the first place, and rightly so. If you followed along in the preceding chapters you should be connecting and interacting comfortably with any computer that you have access to, so file transfer should be a snap.

Suppose you are one of the new breed of "telecommuters" who works at home on a PC and stays in touch with the office through a modem. You could remain connected to the corporate mainframe all day and use its facilities to do your work. But that would be hard on your pocketbook at phone bill time. Or you could prepare your reports, messages, calculations, analyses, and so on, on your PC, and then transmit them to the mainframe when they are done. This approach not only cuts phone costs and keeps your phone from being tied up all day, it also alleviates the load on the mainframe and eliminates the time wasted when phone connections drop unexpectedly and you lose your work.

This is only one scenario among the thousands imaginable. If you are reading this book, you have your own reasons for wanting to transfer files. So let's get on with it.

Transferring Text Files

Begin by using all the familiar steps to start Kermit on the PC: Set the appropriate communication parameters (see Table 7-1), CONNECT, dial the host (other) computer with your modem, and log in with your access codes.

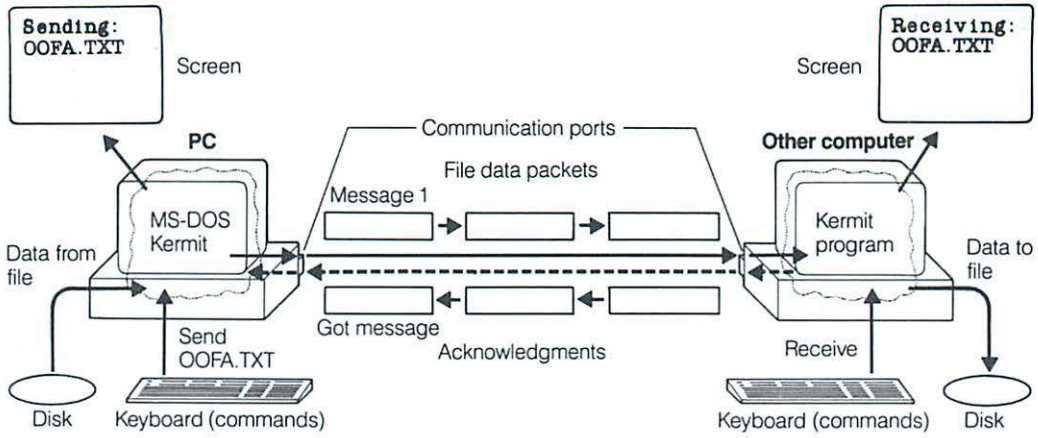


Figure 9-1 Kermit to Kermit

```
C>                                    (The DOS prompt on your hard disk)
C>kermit                              (Start up Kermit on your PC)
IBM PC MS-Kermit V3.0
Copyright (C) Trustees of Columbia University 1982,1990
Type ? or HELP for help
MS-Kermit>connect                     (Begin terminal emulation)

Welcome to VAX/VMS ...

Username: sal                         (Log in)
Password: ______                      (Type password; it doesn't echo)
```

Now comes the new part. To transfer your file, you must run a *second* Kermit program on the host computer. *Both* computers must be running a Kermit program. Otherwise, one side will not know the rules and cannot play the game correctly. One computer has to know how to send the file, and the other must know how to receive it (see Figure 9-1). Luckily, there are Kermit programs available for almost any computer you can think of, and the cost is nominal.

Most host Kermit programs are very similar to MS-DOS Kermit. Usually, you start them by typing `kermit`, and you get a prompt, like `C-Kermit>` or `Kermit-CMS>` or `Kermit-TSO>` or `Kermit-32>`. This helps remind you which Kermit program you are talking to. The basic commands of most Kermit programs are the same: HELP, SET, SEND, RECEIVE, and so forth. Type `help` and read the host Kermit's help message. Type `exit` to quit the host Kermit and return to the host's main system prompt.

```
$ kermit                          (Run VAX/VMS Kermit)

C-Kermit, 5A(100) 8 Feb 90        (The alien Kermit's herald)
Type ? for help                   (and greeting)

C-Kermit>                         (and prompt)
C-Kermit>exit                     (Issue the EXIT command)
$                                 (Back at the VAX prompt)
```

Each different Kermit program has its own peculiarities and documentation to describe its commands, just as MS-DOS Kermit has this book. Some host Kermits (such as C-Kermit for UNIX and VAX/VMS) provide "menu on demand" when you type a question mark, just like MS-DOS Kermit. Certain others, such as IBM mainframe Kermit, cannot do this because the underlying operating environment does not allow it. But almost all Kermit programs have built-in help of some kind.

Take some time to familiarize yourself with the host Kermit program. If you are completely baffled, read the documentation. If you can't find documentation, ask the host system administrator, or contact Kermit Distribution (see the Preface).

In the following examples, you will see different host systems—VAX/VMS, UNIX, and IBM mainframe operating systems like MVS/TSO and VM/CMS. Basic use of Kermit with these systems is the same, differing mainly in the communication parameters you must set to use them successfully. We already saw how to do this in Chapter 7.

The SEND and RECEIVE Commands

The basic commands for file transfer are pretty much what you might expect, SEND and RECEIVE. The SEND command sends the named file or files to the other Kermit program, which must be given a RECEIVE command. The RECEIVE command (you can probably finish this sentence yourself) waits for a file to arrive from the other Kermit, which must be given a SEND command. You have to issue the SEND or RECEIVE command to the remote Kermit first and then escape back (return) to MS-DOS Kermit and issue the corresponding RECEIVE or SEND command.

In the examples, the character for escaping back from the host computer to the PC is shown as *Alt-X*. You can also use *Ctrl-]C*. The two are interchangeable. Use whichever one best matches your typing habits, or use SET KEY to assign this function (`\Kexit`) to any other key of your choice; for example:

```
MS-Kermit>set key \315 \kexit
```

to put it on the F1 key (see Chapter 8).

Uploading Files: The SEND Command

Sending files from your PC to a remote computer is called *uploading*. The procedure for uploading a file from the PC is:

1. Start Kermit on your PC.
2. CONNECT to the remote computer, log in if necessary, and then start Kermit there.
3. Type the RECEIVE command to the host Kermit.
4. Escape back to the PC's `MS-Kermit>` prompt.
5. Type the SEND command, specifying the name of the PC file to be sent.
6. When the transfer is complete, CONNECT back to the host system, and then EXIT from host Kermit.

SEND is one of Kermit's most important commands. It is used so often that it has a special abbreviation, S, even though many of Kermit's other commands also start with S.

Remember to log out from the host when you are finished using it. If you have dialed the host with a modem, logging out should also hang up your phone connection automatically. Watch your modem lights to make sure.[13] If the lights are still on after you log out, your telephone connection is still active. You can hang it up with Kermit's HANGUP command. If all else fails, turn off your modem.

Now let's run through a real example. You have a PC, a hard disk, a Hayes or Hayes-compatible modem,[14] and a Touch-tone telephone, and the remote host is a UNIX system. The file you want to upload from your PC to the UNIX computer is a text file (all printable characters) and is stored on your PC in the directory `\REPORTS` with the filename `REPORT.TXT`. Let's take it from the very beginning, after you have turned on your PC:

```
C>                                  (The DOS prompt on your hard disk)
C>kermit                            (Start up Kermit on your PC)
IBM PC MS-Kermit V3.0
Copyright (C) Trustees of Columbia University 1982,1990
Type ? or HELP for help
```

[13]Most external modems have status lights. Internal modems do not. You can find out the status of an internal modem by using Kermit's SHOW MODEM command.

[14]If you are using a different kind of modem, the dialing procedure and modem responses (like `CONNECT 1200`) may be different. If you are using a "dumb modem," you must dial manually.

```
MS-Kermit>cd \reports              (Change directory to \REPORTS)
MS-Kermit>set speed 1200          (Set speed to match modem)
MS-Kermit>connect                 (Begin terminal emulation)
AT                                (Get attention of Hayes modem)
OK                                (Modem says "I'm ready")
ATDT2125554321                    (Type Hayes dialing command)
rrrriiinnnngggg                   (Phone rings and picks up)
CONNECT 1200                      (The other computer answers)

Ultrix V2.0 (cunixc), tty34
8 users, load average: 0.42 0.42 0.59

login: phyllis                    (Type your username)
password:______                   (Type your password; it won't echo)
$                                 (The UNIX computer prompt)
$ kermit                          (Run UNIX Kermit)

C-Kermit, 5A(100) 8 Feb 90, VAX/Ultrix
Type ? for help

C-Kermit>receive                  (Tell UNIX Kermit to receive a file)
Escape back to your local system and give a SEND command...

Alt-X                             (Hold down the Alt key and press X)
                                  (to escape back to the PC)

MS-Kermit>send report.txt         (Tell PC Kermit to send the file)

      (The file is transferred...)

MS-Kermit>                        (All done; prompt reappears)

MS-Kermit>connect                 (Connect to the UNIX computer again)
C-Kermit>exit                     (Exit from UNIX Kermit)
$ exit                            (Log out from UNIX)
Alt-X                             (Hold down the Alt key and press X)
                                  (to get back to the PC)

MS-Kermit>exit                    (Return to DOS)
```

Congratulations, you have just transferred your first file! You can believe Kermit's `Sending: Complete` message, or you can simply look to see that the filename appears on the receiving computer. If you are still skeptical, you can TYPE or PRINT the newly received file and compare it with the original. And, if you *still* aren't convinced, you can transfer the file back to the PC under a different name and use the DOS command COMP to compare the transferred file with the original file.

Sending Multiple Files

MS-DOS Kermit will also send a group of files with a single command if you include one or more wildcard characters in the SEND command filename; for example:

```
MS-Kermit>send s*.txt
```

sends all files in the current directory whose names start with S and that have a filetype .TXT. MS-DOS Kermit wildcards behave exactly like DOS wildcards, except that you can't use a question mark in the first position of a filename because the question mark gives help in that position; use the # character instead:

```
MS-Kermit>send #of?.txt
```

This command will send all files of type .TXT whose names are exactly four characters long with OF as the second and third characters, such as OOFA.TXT and TOFU.TXT.

File Transfer Display

Notice the display on the screen while the file is being transferred. This gives you an up-to-the-minute status report of the progress of the file transfer:

```
          MS-Kermit: V3.0

          File name: REPORT.TXT
 KBytes transferred: 7
Percent transferred: 35%
            Sending: In progress

  Number of packets: 74
      Packet length: 93
  Number of retries: 0
         Last error: None
       Last message:

X: cancel file, Z: cancel group,
E: exit nicely, C: exit abruptly, Enter: retry
```

The line Sending: In progress will change to Sending: Complete when the file has been transferred successfully. If the file transfer failed for any reason, this line will say Sending: Failed, and the Last error field will display the reason, for instance Insufficient disk space.

The file transfer display fields should be self-explanatory. KBytes transferred means (approximately) how many thousands of characters have been transferred so far. The packet statistics are updated continuously. If you should see them stop, you know something is wrong. Try pressing the Enter key once or twice if this happens.

Several other file transfer display options are available. The command to select them is SET DISPLAY. The options are REGULAR (the display shown above), QUIET (no display at all), and SERIAL (see Chapter 15):

```
MS-Kermit>set display ?  One of the following:
 Quiet, Regular, Serial
MS-Kermit>set display quiet
```

File Transfer Interruption

The bottom line of the file transfer display shows the options you have for interrupting the file transfer:

X Press the X key to stop the current file. The portion of the file transmitted so far is not kept by the receiving Kermit unless it was given the command SET INCOMPLETE KEEP prior to the file transfer. If multiple files are being transferred, the next file will begin.

Z Press the Z key to stop the current file (just like pressing the X key). No more files will be sent, and MS-DOS Kermit will return to its prompt.

E Press the E key if X or Z didn't work.

C Press the C key if E didn't work.

Enter If nothing is happening on the screen and the file transfer seems to be stuck, press the Enter key to wake it up.

To illustrate, suppose you have given the command:

```
MS-Kermit>send *.*
```

to send all of your files. As you sit in your comfy chair watching the file transfer display, you see the filenames change as one file after the other is successfully transferred. But suddenly you see the name HUMONGUS.TXT. This was not one of the files you meant to transfer, and it's so huge that it will take hours for Kermit to finish with it and get on to the next file. This is a good time to use the X key.

Again, suppose you have given the command:

```
MS-Kermit>send *.*
```

This time as you watch the filenames go by, it slowly dawns on you that these are not the files you meant to send. You were CD'd to the wrong directory! Time to bail out: Press the Z key, and Kermit will stop. You will still have to delete whatever unwanted files arrived at the other end.

Sending a File under an Assumed Name

If, for reasons known best to yourself, you want to upload a file from your PC under a filename different from the one stored in the PC, Kermit will let you change the file's name in flight. Just include the new name in the SEND command, after the original name:

```
MS-Kermit>send whoopee.txt serious.doc
```

The PC file WHOOPEE.TXT will be stored at its destination as SERIOUS.DOC.

Downloading Files: The RECEIVE Command

Receiving files at your PC from a remote computer is called *downloading*. The procedure for downloading a file to the PC is as follows:

1. Start Kermit on your PC.
2. CONNECT to the remote computer, log in if necessary, and then start Kermit there.
3. Type the SEND command to the host Kermit, specifying the name of the file to be sent.
4. Escape back to the PC's `MS-Kermit>` prompt.
5. Type the RECEIVE command.
6. When the transfer is complete, CONNECT back to the host system, and then EXIT from host Kermit.

Because the RECEIVE command is so fundamental to Kermit's operation, it can be abbreviated simply R, even though several other Kermit commands start with the same letter.

Like the SEND command, the RECEIVE command can include an optional new name to store the file under when it arrives:

```
C-Kermit>send geo.doc              (Send a file from UNIX)
Alt-X                              (Hold down the Alt key and press)
                                   (the X key to get back to the PC)
MS-Kermit>rec whatsup.doc          (Receive with new name)
```

This will store the UNIX file `geo.doc` as `WHATSUP.DOC` on the PC. You can also specify a device and/or directory to have the file stored there under its own name or under any name you specify:

```
MS-Kermit>receive c:\bugs
```

Most modern versions of Kermit, when sending a file to MS-DOS Kermit, will include advance notice of the file's size. If a file is bigger than the available space on the current disk, MS-DOS Kermit will refuse it. This feature can save you a lot of time and phone charges—imagine having the file transfer fail after receiving 9.9 megabytes of a 10 megabyte file over a long-distance dialup connection at 1200 bps! (You can do the math yourself.)

This advance notice comes in an *attribute packet*, which also may include some other information about the file, such as its creation date. MS-DOS Kermit will use this date when creating the new file on the MS-DOS disk. So if you are a last-minute worker, you cannot pretend you created your file a week ago and Kermit changed its date to today.

If MS-DOS Kermit's treatment of attribute packets bothers you, you can disable it by giving the command SET ATTRIBUTES OFF. Disabling this feature means that no size information will be announced, you will not see what percentage of the file has been received, and received files will be stored with the PC's current date and time.

Now let's try downloading a text file called MEETINGS.TXT from a VAX/VMS computer to the PC. This procedure differs from the previous one only in exchanging the SEND and RECEIVE commands and in the details of using the remote host:

```
C>                                  (The DOS prompt on your hard disk)
C>kermit                            (Start up Kermit on your PC)

IBM PC MS-Kermit V3.0 ...
Copyright (C) Trustees of Columbia University 1982,1990
Type ? or HELP for help

MS-Kermit>set speed 1200            (Set speed to match modem)

AT                                  (Get modem's attention)
OK
ATDT2127654321                      (Type Hayes dialing command)
rrrriiinnnnggggg                    (Phone rings and answers)
CONNECT 1200                        (Modem confirms)

Welcome to VAX/VMS...
Username: roberta                   (Put in your username)
Password: ________                  (Type your password)
$                                   (The VMS system prompt)
$ kermit                            (Run Kermit on the VAX)

C-Kermit, 5A(100) 8 Feb 89, VAX/VMS
Type ? for help

C-Kermit>set def [.reports]         (Change directory)
C-Kermit>send meetings.txt          (Tell VAX Kermit to send the file)

Alt-X                               (Hold down the Alt key and press X)
                                    (to escape back to the PC)

MS-Kermit>receive                   (Tell PC Kermit to receive a file)

      (The file is transferred...)

MS-Kermit>                          (All done; prompt reappears)

MS-Kermit>connect                   (Connect to VMS again)
C-Kermit>exit                       (Exit from VMS Kermit)
$ logout                            (Log out from VMS)
Alt-X                               (Escape back to the PC)

MS-Kermit>exit                      (Return to DOS)
C>
```

That's all there is to it. Now you know how to send and receive files with Kermit. But the book does not end here. That's because neither do the capabilities and options of the MS-DOS Kermit program. You don't have to be a passive receiver. Just like an MS-DOS Kermit receiver, you have options too.

Interrupting File Reception

Suppose that you are receiving files and you want to stop the file transfer. The file interruption keys X, Z, E, C, and Enter work the same way as when sending the file. You have given the command:

```
MS-Kermit>receive
```

and the other Kermit is sending a file or files that you don't want. You pressed the X key, and it didn't work—the file kept coming. So you tried the Z key, but that didn't work either. (This is hypothetical; usually these will work.) At this point, use the E key. That should put a stop to it.

Filename Collisions

Kermit will rename files for you automatically if the name of an incoming file is the same as the name of a file that already exists on the disk and directory to which Kermit is writing the file:

```
          MS-Kermit: V3.0

          File name: MEETING.TXT as MEETING1.TXT
 KBytes transferred: 7
Percent transferred: 35%
            Sending: In progress

  Number of packets: 74
      Packet length: 93
  Number of retries: 0
         Last error: None
       Last message: Renaming file to MEETING1.TXT
```

This safety feature of MS-DOS Kermit is activated automatically. If a file arrives that has the same name as an existing file, the arriving file is assigned a new, unique name so that it won't destroy any existing file. This feature is called *file warning*, and it is in effect unless you turn the feature off. If you really want arriving files to overwrite files of the same name on your PC disk, give the command SET FILE WARNING OFF:

```
MS-Kermit>set file warning off
```

But remember that you did this so that you don't overwrite another file accidentally.

Text versus Binary Files

A *text* file contains all printable letters, numbers, and symbols that you can type and read on the screen. Text files can be transferred between any two Kermit programs. The Kermit programs know how text files are supposed to look on each computer and adjust the format, if necessary, so that these files are always usable after transfer. This way you can type the file, edit it, or whatever else you need to do with a text file as if you had originally created it on the computer it was transferred to. Examples of text files include:

AUTOEXEC.BAT	*Your DOS startup file*
MSKERMIT.INI	*MS-DOS Kermit's initialization file*
KERMIT.HLP	*Help file for MS-DOS Kermit*

A *binary* file contains information intended for the computer, not for you. Binary files are specific to a particular application or a particular kind of computer or device. Not only must you tell Kermit that the file should not be changed during file transfer, but usually you will not be able to use this file on a different type of computer. You may still want to do binary transfer to store programs on a system that cannot use them so that others can download them at a later time. Examples of binary files include:

COMMAND.COM	*The MS-DOS command processor*
KERMIT.EXE	*The MS-DOS Kermit program*
BUDGET.WKS	*A Lotus 1-2-3 spreadsheet*
PHONES.DBF	*A dBase database file*

With the sophistication of word processors, the distinction between text and binary is not always as clear as it seems. A word processor program that allows you to save a file with underlining and **bold** headings, for example, has inserted some control information in the file, as well as the text you typed, so that when you view or print the file, it can remember where to underline and boldface.

Since this control information is not text, the file must be categorized as binary to be transferred correctly. If this type of file is transferred to another type of computer, it cannot be used there in any normal way because the other computer won't know what to do with the control characters that were inserted. If there is a way for you to save the file as "text-only" or "ASCII-only" (most word processors let you do this), all the special formatting controls are discarded, and you should be able to transfer this file in usable form to another computer, but without the underline, **boldface**, *italics*, and other special effects.

In general, neither the PC nor Kermit can tell whether a file is text or binary. Only you (and your hairdresser) know for sure.

Transferring Binary Files

Transferring binary files is the same as transferring text files except that you have to give the command SET FILE TYPE BINARY to *both* Kermit programs.

Remember that unless you are transferring a binary file between two computers that are of the same type (like two IBM PCs), you usually cannot use the binary file on the computer you send it to. But if you transfer a binary file from the IBM PC to, say, an IBM mainframe and then from the IBM mainframe to another IBM PC, you can use the file on the other IBM PC as long as you remember to SET FILE TYPE BINARY each step of the way.

```
MS-Kermit>connect                    (Connect to the host computer)
.kermit                              (Start Kermit on the host)
Kermit-CMS>set file type binary      (The important command)
Kermit-CMS>receive                   (Tell it to receive)
Alt-X                                (Hold down Alt key and press X)
MS-Kermit>set file type binary       (Here too!)
MS-Kermit>send kermit.exe            (Send the binary file)

      (The file is transferred...)

MS-Kermit>                           (Finished; prompt reappears)
```

Unless you are transferring files between two PCs, *don't mix text and binary files* in the same SEND command. Most Kermit programs can be in only one file mode at a time, text or binary.

Transferring Files with IBM Mainframes

When transferring files with IBM mainframes, make sure to set the appropriate communication parameters before connecting (see Chapter 7). To recap, if you have a *fullscreen* connection, you can usually use all of MS-DOS Kermit's default parameters, except you probably have to use some kind of parity, and you have to set your terminal type appropriately:

```
MS-Kermit>set parity even            (Use even parity)
MS-Kermit>set terminal vt102         (Set terminal type)
```

For a *linemode* connection (line at a time, no screen control), you have to change most of MS-DOS Kermit's defaults:

```
MS-Kermit>set parity mark            (This one may vary)
MS-Kermit>set flow none
MS-Kermit>set handshake xon          (This one too)
MS-Kermit>set duplex half
```

Aside from the different communication parameter settings, you can use Kermit with an IBM mainframe just as you would any other kind of host.

Trouble

Maybe "it's not easy being green," but it should be easy to transfer files with Kermit. And we will see that it will get even easier. The hard part is figuring out what you did wrong when file transfer doesn't work.

First, did you set all the communication parameters correctly before connecting to the other computer? You look confused. Did you skip the earlier chapters? Tsk tsk. Go back and read them (especially Chapter 7) to see how you were supposed to prepare for file transfer. Next, issue the MS-DOS Kermit SHOW COMMUNICATIONS command, and look for any obviously incorrect settings (parity, flow control, handshake). Fix them with SET commands, and then try again.

If you are able to connect successfully, but file transfer still doesn't work, consider:

The Obvious

Did you remember to start Kermit on the other computer? Did you remember to give it a SEND or RECEIVE command before escaping back to the PC?

Parity

The most common cause of file transfer failure is parity. Kermit would prefer not to use parity, but if the host computer or the network between your PC and the host uses it, you must tell Kermit about it; otherwise, the unexpected addition of parity bits to the characters in the Kermit packet will cause Kermit's error checking to fail. Solution: SET PARITY EVEN (or whatever the parity really is). Give this command to *both* Kermit programs.

Duplex

If you are connected to a half-duplex computer system, you should have given the command SET DUPLEX HALF. But this command has several functions and you might not want them all: It enables local echo (equivalent to SET LOCAL-ECHO ON), it turns off full-duplex flow control (equivalent to SET FLOW NONE), and it enables a kind of hardware communication line access control called RTS/CTS (see the Glossary). It is possible that RTS/CTS is causing the problem. To find out, try this:

```
MS-Kermit>set duplex full
MS-Kermit>set local-echo on
MS-Kermit>set flow none
```

Handshake

For transferring files with half-duplex computer systems, the normal line turnaround handshake character is XON. Did you give the command SET HANDSHAKE XON? Your host computer may use a different character, or none at all, for this purpose. Consult your host system documentation or administrator to find out, or experiment with different values for SET HANDSHAKE.

Adapting to Communication Line Noise

Suppose you have a direct, high-speed connection to your company's mainframe computer at work, but from home you use your modem and regular telephone lines to access the same computer. The regular telephone lines are prone to static or clicking noises, but the direct line is far more "clean"—free from this kind of interference.

When a connection is more apt to have noise interference, there are a few parameters in the MS-DOS Kermit program that can help prevent file transfers from failing:

SET RECEIVE PACKET-LENGTH

The messages into which Kermit breaks up your file are called packets, normally 94 characters long. When the connection is noisy, reducing the number of characters sent in a single shot increases the chance that they will all make it to the other side intact and decreases the cost of packet retransmission.

SET RETRY

The Kermit program will try to send a particular packet a certain number of times (five, for example) before it gives up. This is how Kermit detects a broken connection. But under noisy conditions, the fact that a packet has been retried many times need not mean the connection has dropped. Under these conditions, you can increase this number to prevent unnecessary failures.

SET BLOCK-CHECK

No error detection mechanism is foolproof. Under noisy conditions, it's more likely (but still *very* unlikely) that a transmission error could slip through undetected. To decrease the chance of this happening, you can select a more powerful detection method. There are three levels: 1, 2, and 3. The higher the level, the stronger the error detection, but also the higher the price paid in transmission overhead. If you do not say otherwise, 1 is used.

In the following example, we instruct Kermit to be more persistent under noisy conditions by allowing more retries, using smaller packets, and using the strongest of Kermit's error-checking techniques:

```
MS-Kermit>set retry 20
MS-Kermit>set packet-length 40
MS-Kermit>set block-check 3
```

Improving Kermit's Performance

Let us hope that by now you have Kermit transferring files correctly and completely. If you had problems, you've very likely solved them. It may have taken some experimentation and even some reading, but the fact is that Kermit can transfer files in almost any environment with almost any kind of computer—a claim that no other file transfer protocol

can make. But the price for this universality is paid in efficiency. Now let's look at some ways of making file transfer go faster.

Packet Length

One factor in Kermit's success is its use of relatively short packets: The normal packet length is 94. These short packets are able to slip through all sorts of narrow openings in which longer packets would get stuck. But if you know that your connection will allow long bursts of data, you can increase Kermit's packet length to any number up to 2000. The longer the packet, the higher the ratio of real information to protocol overhead, and therefore the greater the efficiency of the file transfer. But to use long packets, *both* Kermit programs must support this feature.

The command that controls the packet length is SET RECEIVE PACKET-LENGTH. This is the very same command that helped us adjust to noisy and clean communication lines by making the packets smaller. The Kermit program that is receiving packets controls the packet length, so give this command to MS-DOS Kermit if you are downloading, and give it to the host Kermit if you are uploading. If you plan to send files in both directions, give it to both Kermits:

```
MS-Kermit>set receive packet-length 1000
MS-Kermit>connect
C-Kermit>set receive packet-length 1000
C-Kermit>
```

(Remember, Kermit commands can be abbreviated. Normally, all you would have to type here is SET REC PAC 1000.) When you transfer a file using long packets, watch the file transfer display. The `Number of packets` field will change more slowly because there is more data in each packet, as you can see by looking at the `Packet length` field:

```
          MS-Kermit: V3.0

          File name: REPORT.TXT
 KBytes transferred: 7
Percent transferred: 35%
            Sending: In progress

  Number of packets: 7
      Packet length: 1000
  Number of retries: 0
         Last error: None
       Last message:
```

A big advantage of long packets is that they can be used on either full-duplex or half-duplex connections. But there are also several risks. First, long packets may trigger the very problem that Kermit's regular short packets were designed to avoid: buffer overflows and the resulting data loss. If 94-character packets work for you, but 1000-character packets consistently fail, try to home in on the largest size that works.

Second, Kermit's normal error-checking technique is really not strong enough for very long packets. So when using long packets, you should also SET BLOCK-CHECK 2 or 3.

Third, if the connection is noisy, long packets may actually *reduce* your efficiency rather than increase it. That's because the longer the packet, the greater the chance it will be damaged under noisy conditions, and the more time it takes to retransmit it.

You can use Kermit's SHOW STATISTICS command to measure your file transfer efficiency. Using the same transmission speed, compare the file characters per second for long packets with the file characters per second for regular packets.

Sliding Windows

Normally, the sending Kermit program transmits a packet (message) and then waits for the receiving Kermit program to send a response back saying "I got it." The sending Kermit program then transmits another packet. Kermit's "stop-and-wait" style of packet exchange works on both full- and half-duplex connections. But stopping and waiting for a reply can be costly on long-distance connections, like those through earth satellites or public data networks.

On full-duplex connections, where both computers can send and receive data simultaneously, it is not necessary for each Kermit to be silent while the other one is transmitting. The delay caused by waiting for a reply to each packet can be eliminated if we allow a certain number of packets to be sent without replies and let the replies come later. This "certain number" is called the *window size*. As long as the first reply comes before the window size is exceeded, transmission of packets can be continuous, even over long-delay connections. Kermit's window size can be from 1 to 31:

```
MS-Kermit>set window 8
```

It is best to use windows with regular (short) packet lengths of 94 or fewer characters. This way, loss of performance is minimized when packets are damaged by noise. On a very clean connection, however, you can combine sliding windows and long packets:

```
MS-Kermit>set window 8
MS-Kermit>set receive packet-length 250
```

The window size (number) you request is the maximum number Kermit will use. The actual number of window slots in use at any instant shows up in the file transfer display:

```
          MS-Kermit: V3.0

          File name: REPORT.TXT
 KBytes transferred: 14
Percent transferred: 70%
            Sending: In progress
Window slots in use: 3
```

```
Number of packets: 140
    Packet length: 94
Number of retries: 0
       Last error: None
     Last message:
```

This number is a determined by various factors, including the amount of delay from one end to the other and the amount of noise on the connection. The longer the delay, or the worse the noise, the more window slots need to be used.

Sliding windows can be used only with other Kermit programs that support this feature, such as PRIME Kermit, or C-Kermit 5A or later on UNIX and VAX/VMS. If you try to use sliding windows with a Kermit program that does not have this feature, the two Kermits will automatically use the stop-and-wait method instead.

Chapter 10

Using a Kermit Server

Now that you are an experienced Kermit user, you are ready to move on to the next level of commands. No longer must you continuously escape back and forth between MS-DOS Kermit on the PC and the Kermit program on the other computer if you are transferring a lot of files whose names are not similar enough for wildcards or if you are sending some files and receiving some others. Instead, you can connect to the other computer and start up the Kermit program there as before, but this time you can put the host Kermit into *server mode* by issuing the SERVER command.

Once the remote Kermit is in server mode, you only need to type commands to the PC's `MS-Kermit>` prompt to send and get files. MS-DOS Kermit will relay any commands intended for the other computer to the remote Kermit server automatically. In the following example, we use a Hayes (or Hayes-compatible) modem to dial up an IBM mainframe, set the mainframe Kermit to be a server, and transfer files to the PC and from the PC.

```
C>                                  (The DOS prompt)
C>kermit                            (Start up Kermit on your PC)

IBM PC MS-Kermit V3.0 ...
Type ? or HELP for help

MS-Kermit>ibm                       (See Chapter 14)
MS-Kermit>set speed 1200            (Set the speed)

AATT                                (Half duplex, double echo)
OK                                  (Modem is ready)
ATDT2127654321                      (Issue the dial command)
```

```
rrrriiinnnnggggg                    (Phone rings and answers)
CONNECT 1200                        (Modem confirms)

VIRTUAL MACHINE/SYSTEM PRODUCT
!
.login louie                        (Type your username)
Enter password: _______             (Put in your password)
.                                   (The VM/CMS prompt is a period)
Ready; T=0.07/0.11 11:58:28
CMS
.kermit                             (Run mainframe Kermit)
Kermit-CMS Version 4.1 (10 Mar 89)
Enter ? for a list of valid commands

Kermit-CMS>server                   (Put it in server mode)
Entering server mode.  Please escape to
local Kermit now.  To terminate the server
use the BYE or FINISH commands.

Alt-X                               (Escape back to your PC)

MS-Kermit>send oofa.*               (Send all of my OOFA files)

    (The files are transferred...)

MS-Kermit>get meeting.txt           (Tell server to send a file)

    (The file is transferred...)

MS-Kermit>                          (Done; prompt reappears)
MS-Kermit>connect                   (Go back to the IBM mainframe)
```

As you can see, you can send and receive as many files as you like without having to escape back and reconnect.

If you do reconnect, you will see the screen as you left it when you last escaped back to MS-DOS Kermit. When you try to type to the IBM mainframe directly, nothing happens. This is because it is still in server mode. You can communicate with it only by typing commands in response to the `MS-Kermit>` prompt.

When you are done, you can give the BYE command. This shuts down the remote Kermit server and logs out your host session. In this case, there is no need to reconnect.

```
MS-Kermit>bye                       (Terminate mainframe session)
C>                                  (BYE also exits to DOS)
```

If you want to shut down the server and log out the host session but remain in MS-DOS Kermit, use the LOGOUT command instead:

```
MS-Kermit>logout                    (Terminate mainframe session)
MS-Kermit>                          (Back to MS-Kermit prompt)
```

If you want to continue terminal emulation with the remote computer, give the FINISH command and reconnect:

```
MS-Kermit>finish                    (Terminate mainframe session)
MS-Kermit>connect                   (Go back to remote computer)
Kermit-CMS>exit                     (Exit from remote Kermit)
                                    (Back at mainframe prompt)
```

Now you can do further work on the mainframe. Remember to log out when you are finished.

Summary of Commands to Use with Kermit Servers

All the following commands can be interrupted by pressing the X, Z, C, or E key in the same way as when using Kermit to SEND or RECEIVE.

SEND

The SEND command is the same as before. It tells MS-DOS Kermit to deliver a file or file group to the server. You can also include a different name to send a file under:

```
MS-Kermit>send boring.txt exciting.txt
```

GET

The GET command sends a special command to the server, telling it the name of the file you want it to send to your PC:

```
MS-Kermit>get mail.txt
```

When communicating with a Kermit server, you must use the GET command rather than the RECEIVE command. If you use RECEIVE with a server, the server will not know what to do, and MS-DOS Kermit will wait a long time for a file that is not going to come. If you get into this situation, press the C key to get the Kermit prompt back immediately. You can also specify a new name for the file when it arrives at your PC by typing GET alone on a line. You will be prompted separately for the two names:

```
MS-Kermit>get
 Remote Source File: profile exec
 Local Destination File: profile1
```

If you change your mind at one of the filename prompts and decide you don't want to issue the GET command after all, type *Ctrl-C* (hold down the Ctrl key and press the letter C) to get the `MS-Kermit>` prompt back.

FINISH

The FINISH command tells the server to get out of server mode and return to its interactive Kermit prompt so that you can connect back to the host and do more work.

BYE

The BYE command tells the remote Kermit server to terminate itself and your entire session on the host. If you give the BYE command, you don't have to (and probably cannot) connect back to the host and log out. The BYE command also causes MS-DOS Kermit to exit and return to DOS.

LOGOUT

The LOGOUT command is exactly like BYE, but it does not exit from MS-DOS Kermit.

The Server's Remote File Services

But wait, there's more! Besides sending and receiving files, the remote Kermit server offers a selection of file management and host access functions, all accessible from your PC's MS-Kermit> prompt. The key to these services is the REMOTE command. You can give the REMOTE commands as shown, and the results will be displayed on your screen. If you want the results to go to a file on your PC, use the DOS redirection symbol, >, followed by a DOS filename:

```
MS-Kermit>remote directory x*.* > \lucy\dir.txt
```

REMOTE CD

Changes the working directory on the host. You should specify the name of the directory to change to. If you give this command without including a directory name, the server will change to your default directory on the remote host:

```
MS-Kermit>rem cd /usr/jrd
/usr/jrd
MS-Kermit>rem cd
/usr/fdc
MS-Kermit>
```

The new directory will be the directory used for all file operations on the host unless you specify otherwise. If a password is required, add the password on the same line:

```
MS-Kermit>rem cd ps:<sy.fdc> secret
```

REMOTE DELETE

Asks the server to delete the specified files on the remote host computer:

```
MS-Kermit>remo del *.obj
MS-Kermit>remo del margaret/*.*
MS-Kermit>
```

You can delete only those remote files for which you have write or delete permission.

REMOTE DIRECTORY

Lists files on the remote computer. If you don't include a directory or file specification, all files in the current directory will be listed. If you do include a directory or file

specification, only the files that match will be listed:

```
MS-Kermit>remo dir o*.*
total 2
-rw-rw---- spg  224 Feb  8 15:02 oofa
-rw-rw---- fdc  115 Nov 10 15:02 oofa.c
-rw-rw---- spg 1287 Jun 11 15:02 oofa.sh
MS-Kermit>
```

REMOTE HELP

Asks the server to list the services it provides. Kermit program services may vary. Here, for example, is the response from the VAX/VMS C-Kermit server:

```
MS-Kermit>remote help
C-Kermit Server REMOTE Commands:

GET files  REMOTE CD [dir]     REMOTE DIRECTORY [files]
SEND files REMOTE SPACE [dir]  REMOTE HOST command
MAIL files REMOTE DELETE files REMOTE WHO [user]
BYE        REMOTE PRINT files  REMOTE TYPE files
FINISH     REMOTE HELP
```

This tells you which commands the server can execute for you, for example:

```
MS-Kermit>remote space [catherine]
```

If you type an MS-DOS Kermit server-related command that is not on the server's list, you will get an error message:

```
MS-Kermit>remote message Guess who?
Error: Unimplemented server function
MS-Kermit>
```

The REMOTE commands (except for REMOTE HOST) are generic in that they are the same (if the program supports them) no matter which Kermit server you are connected to. For example, Kermit programs on the IBM mainframe, UNIX, VAX/VMS, and PDP-11 all use the command REMOTE TYPE to type a file, REMOTE DIRECTORY to list filenames, and so on, even though each of these host computers might use different commands for the same requests if you were talking to them directly.

REMOTE HOST

The REMOTE HOST command, however, is specific to each kind of host. With this command, you tell MS-DOS Kermit to ask the host Kermit server to ask the host operating system to execute the given command, written in the host system's own command language. For example, here is how you would use the REMOTE HOST command to ask the Kermit server to rename a file on various kinds of hosts (MS-DOS Kermit currently does not have a REMOTE RENAME command):

```
MS-Kermit>rem host mv oofa.new oofa.old          (UNIX)
MS-Kermit>rem host rename oofa.new oofa.old      (VAX/VMS)
MS-Kermit>rem host rename oofa new a oofa old a  (VM/CMS)
```

The command that you ask the remote host to execute *cannot be an interactive command* that requires a response from the user.

REMOTE KERMIT

Sends a command to the remote Kermit server in its own command language, for example:

```
MS-Kermit>remote kermit set block 2
```

REMOTE LOGIN

Logs in to a remote Kermit server that has been set up to require a username and password:

```
MS-Kermit>rem login leslie anything
```

Here, "leslie" is the username, and "anything" is the password. If username and password information is omitted from the command line, you will be prompted for them. If the password is given on the REMOTE LOGIN command line, it will echo, but if you're prompted for it, it will not echo:

```
MS-Kermit>rem login ken secret
MS-Kermit>rem login
 Username: ken
 Password: ______
 Account:
```

At the `Account:` prompt, type your account, if one is required, or just press the Enter key.

REMOTE MESSAGE

Sends a one-line message for display by the remote Kermit server. This command is useful when the remote Kermit server is a PC with a person looking at the screen, for example:

```
MS-Kermit>rem mess OK, I'm finished...
MS-Kermit>rem mess You can sleep now!
```

REMOTE SET

Changes one of the remote Kermit server's settings, for example:

```
MS-Kermit>rem set file type binary
```

REMOTE SPACE

Displays the amount of disk space available on the remote host so that you can estimate if it has enough room for the files you plan to send. You can specify a device or directory name, or you can leave it out to get a report about the current device and directory:

```
MS-Kermit>remote space
MS-Kermit>rem spa $disk1:[donna]
```

REMOTE TYPE

Displays a remote file on your PC screen:

```
MS-Kermit>remote type max.txt
```

REMOTE WHO

Lists the users who are logged on the remote system or information about the specified user:

```
MS-Kermit>remote who              (All users)
MS-Kermit>remote who anne         (Specific user)
```

The results depend on the host operating system.

MS-DOS Kermit also has a special command called MAIL. It is just like the SEND command and can be used in conjunction with either a Kermit server or a Kermit program that has been given the RECEIVE command *if* the host Kermit supports this feature. The MAIL command causes MS-DOS Kermit to send the specified file, but instead of storing it on the remote computer's disk, the remote Kermit program sends it as electronic mail to the specified user:

```
MS-Kermit>mail oofa.txt frank
```

MS-DOS Kermit is able to take advantage of most of the functions offered by the popular Kermit servers. You will find this mode of operation most convenient if you need to transfer collections of files in both directions while checking your directory on the host, deleting files there, and the like. If you transfer only the occasional file, then the SEND/RECEIVE style of operation may be more convenient.

Chapter 11

Making Your PC the Remote Computer

MS-DOS Kermit is normally used to *initiate* a connection to a remote host or service. But a PC with Kermit can also act as a remote host itself and receive calls from other computers. There are two ways to do this.

Method 1: Server Mode

The safer and more secure method of remote operation is MS-DOS Kermit's server mode. To use the server option, start MS-DOS Kermit, and then set the desired speed and other parameters as you've done before. Then type the SERVER command. If your PC is connected to a modem, you must also put the modem in *answer mode*. On a Hayes or Hayes-compatible modem, the command is ATS0=1 (that is a zero, not the letter O):

```
C>kermit                        (Run Kermit)
MS-Kermit>set speed 2400        (Set dialup speed)
MS-Kermit>connect               (Connect to the modem)
ATS0=1                          (Put modem in answer mode)
Alt-X                           (Escape back)
MS-Kermit>server                (Enter server mode)
```

The SERVER command can also accept an operand that tells how many seconds to remain in server mode before returning to the `MS-Kermit>` prompt:

```
MS-Kermit>server 3600           (Serve for 1 hour)
```

or until what time to run in server mode:

```
MS-Kermit>server 22:00:00       (Serve till 10:00 P.M.)
```

This way, you can tell your friend "Call my PC between 9:00 and 10:00 P.M. and get the OOFA file." But if your friend forgets to call, your PC won't be sitting in server mode indefinitely. In fact, using this mechanism you can schedule your PC to automatically run different jobs at different times of day. To find out how to start the server (or any other Kermit operation) at a specified time, read about the PAUSE command in Chapter 14.

Remote Commands

The MS-DOS Kermit server supports the following commands from the client Kermit:

SEND	REMOTE CD	REMOTE LOGIN
GET	REMOTE DELETE	REMOTE MESSAGE
FINISH	REMOTE DIR	REMOTE SEND
BYE	REMOTE HELP	REMOTE SET
LOGOUT	REMOTE HOST	REMOTE SPACE
	REMOTE KERMIT SET	REMOTE TYPE

These commands behave in the normal way; for example, to get a directory listing from the remote server's disk:

```
MS-Kermit>rem dir oofa          (Ask for list of remote files)
 Volume in drive C is BENICE
 Directory of  C:\TAKA

OOFA     C        6583   3-31-88  10:50a
OOFA     EXE      8335   3-31-88  10:50a
OOFA     HLP       655   7-30-89   5:51p
       3 File(s)   1445888 bytes free
MS-Kermit>
```

The REMOTE HOST command lets you run any DOS command on the server PC, displaying the output on your screen. The following example gets a listing of all the subdirectories of the current remote directory:

```
MS-Kermit>rem host dir | find "<DIR>"
SPSG          <DIR>       2-08-89   4:03p
SRSKM         <DIR>      11-10-89  11:32a
MS-Kermit>
```

Be careful not to invoke *interactive* remote PC commands like the line editor EDLIN. There is no way to carry on a dialog with the remote application through a Kermit server. If you need to do that, redirect the console with the DOS CTTY command as described in the section on the second method of remote operation, "Redirecting the DOS Session."

Security Features

The MS-DOS Kermit server allows users to change directories, read files, create files, and even delete files, as well as to run any DOS command. If you don't want to expose your PC to that kind of risk, you can have Kermit disable selected server functions using the DISABLE command:

DISABLE CD
(or CWD) Entirely disables changing of directories.

DISABLE DELETE
Confines deletion of files to the current directory.

DISABLE DIRECTORY
Confines production of directory listings to the current directory.

DISABLE FINISH
Entirely disables shutting down the server (applies also to BYE).

DISABLE GET
Confines getting files from the server to the current directory.

DISABLE HOST
Entirely disables execution of all REMOTE HOST (DOS) commands.

DISABLE KERMIT
Disallows use of the REMOTE KERMIT command.

DISABLE LOGIN
The REMOTE LOGIN command is not required.

DISABLE SEND
Forces files sent to the server into the current directory.

DISABLE SPACE
Disables asking the server for a disk space report with the REMOTE SPACE command.

DISABLE TYPE
Confines the REMOTE TYPE command to the files in the current directory.

DISABLE ALL
All the above.

Certain functions are disabled entirely, whereas others are confined to the current directory. A disabled function can be turned back on using the ENABLE command, which allows you to specify all the same functions as DISABLE, for example:

```
MS-Kermit>enable login
```

You can obtain an extra level of security by setting a username and a password for server access:

```
MS-Kermit>set server login name password
MS-Kermit>server
```

This requires the user, after establishing the dialup connection, to give the command:

```
MS-Kermit>remote login name password
```

If the server has been set up to require a login, and you send any other command to it before logging in, it will respond with a message to the effect that login is required:

```
The server PC is set with the following security:
MS-Kermit>set prompt MS-Server>
MS-Server>disable all
MS-Server>enable login
MS-Server>set server login linda secret
MS-Server>server

Someone tries to access this PC without access codes:
MS-Kermit>get oofa.txt
REMOTE LOGIN is required

Someone tries to access this PC with wrong access codes:
MS-Kermit>remote login raynette alf
Invalid login information

Someone accesses the PC with correct access codes:
MS-Kermit>remote login linda secret
MS-Kermit>
```

Disengaging from the Server

The user can terminate the server session on your PC by using the BYE or LOGOUT commands described in Chapter 10. If you want someone else to be able to call your PC server, start it this way:

```
MS-Kermit>disable finish
MS-Kermit>server
```

Then, if a user gives a BYE or LOGOUT command, the server will remain active and waiting for the next user to call.

Method 2: Redirecting the DOS Session

This second method of obtaining remote access to the PC is not recommended, but since it is sometimes used, it should be discussed.

The DOS command CTTY can be used to redirect a DOS session from the normal console (keyboard and screen) to a communication port:

```
C>ctty com1
```

After this command is issued, the DOS command prompt will move to COM1, and DOS will accept commands only from COM1, not from the keyboard. Now other computers or terminals can access your PC by connecting to (or dialing up) your COM1 port. If you have a modem attached to your PC so that people can dial up, you must put it in answer mode.

When accessing DOS through the communication port, you can use only "well-behaved" DOS-level character-mode applications like EDLIN. Any application that puts graphics on the screen or that requires Alt or function (F) keys, cannot be used. If you start such an application, you'll be locked out. Even well-behaved applications can lock you out if they get an error because the familiar DOS error message:

```
Abort, Retry, Fail?
```

appears on the real screen and expects a reply on the real keyboard.

If you understand the limitations and risks of using DOS in this manner, you can make effective use of it within this framework. Let's say you want to be able to access your office PC from your home PC and that you have Hayes modems for each PC. Here is the last thing you would do before leaving work:

```
C>kermit                        (Run Kermit at work)
MS-Kermit>set speed 2400        (Set dialup speed)
MS-Kermit>connect               (Connect to the modem)
AT                              (Make sure modem is working)
OK                              (It is)
ats0=1                          (Put it in answer mode)
Alt-X                           (Escape back to Kermit)
MS-Kermit>exit                  (Exit from Kermit)
C>ctty com1                     (Now move DOS to COM1)
```

When you get home (after eating dinner), you can dial up your office PC:

```
A>kermit                        (Run Kermit at home)
MS-Kermit>set speed 2400        (Set dialing speed)
MS-Kermit>connect               (Connect to your home modem)
AT                              (Make sure it's working)
OK                              (It is)
atdt7654321                     (Call your office number)
CONNECT 2400                    (Hayes says connection is made)
                                (Press Enter key)
C>                              (DOS prompt from office PC)
```

When you are dialing a remote PC from another PC, both Kermit programs will have the same prompt, which can be confusing. You can use Kermit's SET PROMPT command to give distinct prompts to the two programs so that you'll always know which one you're talking to:

```
MS-Kermit>set prompt Work>      (This one is at the office)
Work>                           (Here's the new prompt)
Alt-X                           (Escape back to home PC)
MS-Kermit>set prompt Home>      (This one is at home)
Home>                           (New prompt on home PC)
```

Before you can transfer files, you have to let the Kermit program on your work PC know that it is running on COM1 rather than on the keyboard and screen because DOS has no way of letting Kermit know this. Unless you tell it, the remote Kermit's file transfer display will be mixed up with the file data that is being transferred, and this may prevent file transfer from working. The command to use is SET REMOTE:

```
Home>connect                    (Connect to the remote PC)
                                (Press Enter key)
Work>set remote on              (Tell remote Kermit it's remote)
```

Now you're ready to transfer files:

```
Work>receive                    (I'd better send OOFA.TXT)
Alt-X                           (Escape back to home)
Home>send oofa.txt              (No PC is complete without it)
```

To restore your work PC to normal, do this:

```
Work>exit                       (Exit from remote PC Kermit)
C>ctty con                      (Put its console back to normal)
Alt-X                           (Escape back to home PC)
Home>hangup                     (Hang up the phone)
Home>exit                       (And exit back to DOS)
A>
```

CTTY CON directs DOS back to the real keyboard and screen on your PC at work. You cannot do this from the work PC's keyboard except by rebooting the PC with *Ctrl-Alt-Del* (press the Ctrl, Alt, and Del keys simultaneously).

Aside from the operational risks of remote DOS operation, there is a security risk. Anybody who knows your phone number can call your PC and wreak havoc with your files. DOS has no built-in security features like user IDs or passwords to regulate access. This is why it's important to put your work PC back to normal when you're finished using it. That way, if anyone else dials it up, they won't be able to communicate with it at all.

Chapter 12

Transferring Files without the Kermit Protocol

Now that you know how to transfer files with another computer that has a Kermit program, it's time to consider the unthinkable: What if the other computer *doesn't have Kermit?*

You have several courses of action:

- Badger the computer's system manager mercilessly until Kermit is installed.
- Get Kermit and install it yourself.
- If a Kermit program doesn't exist for a particular computer (there are still a few), write one!
- Use MS-DOS Kermit to transfer files without error correction.

This chapter will discuss the last alternative. You already know how to download a file without error correction. If you don't believe this, review the section on session logging in Chapter 8. How can you use session logging to download a file? Simple: Just display the file on the remote host while logging the session. For uploading files to computers that don't have Kermit, there is a special command, TRANSMIT.

But (you may ask) if that's all there is to it, what do I need the Kermit protocol for? Here are just a few reasons: You can't transfer binary files this way, you can't transfer more than one file at a time, transmission errors and data loss can't be detected, filenames and

other characteristics are not transmitted, and the whole process is cumbersome and unreliable. But it *is* better than nothing.

For both uploading and downloading, you must first get connected and logged in, setting all the appropriate communication parameters, with particular attention to duplex, flow control, and handshake.

Downloading a Host File to the PC

The trick here is to type the host command to display the desired file *up to but not including the Enter key*, which actually starts the command. Then escape back to MS-DOS Kermit, start the session log, connect back to the host, and press the Enter key to start the command. When you are done, escape back again and close the session log.

Let's look at an example. Here we log in to a UNIX system, with which Kermit can use its default parameters and where the command to display a file is "cat" (yes, cat).

```
C>kermit                          (Start Kermit on the PC)

MS-Kermit>connect                 (Begin terminal emulation)
                                  (Press the Enter key)
login: marilyn                    (Log in)
Password: ________                (Type your password)

$ cat resume.txt                  (Type the command to display the)
                                  (file, but don't press Enter yet!)

Alt-X                             (Hold down the Alt key and press X)
                                  (to escape back to MS-DOS Kermit)

MS-Kermit>log sess resume.hlp     (Start the session log)
MS-Kermit>connect                 (Go back to the host)
                                  (Now press Enter key)
```

As the host file is displayed on your screen, it is also recorded in the PC file with the name you specified in the LOG SESSION command.

This process does not terminate by itself. You have to watch. When you see the end of the file and the host computer prompt appears again, escape back to MS-DOS Kermit and close the session log:

```
(many lines are displayed...education, etc.)
  References will be provided upon request.
$                                  (Host computer prompt appears)
Alt-X                              (Hold down the Alt key and press X)
MS-Kermit>close session            (Close the session log)
```

A copy of the file is now on your DOS disk. It is almost guaranteed to be somewhat different from the original copy, if only because it has the host prompt at the end. You may also find terminal messages mixed in, gaps, or unexpected characters caused by interference. If you want the file to be clean and pure, you have to go over it carefully with a text editor on your PC.

Uploading a PC File to the Host

To send a file to a host that doesn't have Kermit, you can use a combination of MS-DOS Kermit's TRANSMIT command and the host's file-creation mechanism.

Kermit's TRANSMIT command works only with text files. It sends the file as if you were typing it, a line at a time. Full-duplex flow control or half-duplex handshake is used to make sure that Kermit does not send the text faster than the host can store it, but there is no provision at all for error detection or correction.

On the host end, you should do whatever you normally do to create a new file whose text is entered from the terminal. In many cases, this means starting a line-oriented text editor and putting it in text insertion mode. Some hosts have commands that let you type text directly from the keyboard into a file. Table 12-1 shows some examples (for creating a file called `oofa.txt`).

Table 12-1 File Creation Commands

System	*To Begin*	*To End*
UNIX	`cat > oofa.txt`	*Ctrl-D*
VAX/VMS	`copy tt: oofa.txt`	*Ctrl-Z*
MS-DOS	`copy con oofa.txt`	*Ctrl-Z*, *Enter*
Most Others	(Use a text editor)	

Let's try this with a VAX/VMS system:

```
C>kermit                          (Start Kermit on the PC)

MS-Kermit>connect                 (Begin terminal emulation)
                                  (Press the Enter key)
Username: mary                    (Log in)
Password: _________               (Type your password)

$ copy tt: oofa.txt               (Copy terminal to file)

Alt-X                             (Hold down the Alt key and press X)
MS-Kermit>transmit oofa.txt       (Send the file)
MS-Kermit>connect                 (Go back to the host)
Ctrl-Z                            (Hold down the Ctrl key and)
                                  (press Z to close the file)
Exit                              (VMS confirms the file is closed)
$ dir /size/date oofa.txt         (Is it really there?)

Directory $DISK1:[MARY]

OOFA.TXT;1    2  8-FEB-1990 11:32:29.75

Total of 1 file, 2 blocks.

$ logout                          (Remember to log out)
Alt-X                             (Hold down the Alt key and press X)
MS-Kermit>                        (Back at MS-Kermit> prompt)
```

The file (or pieces of it!) is now on your VMS disk. If you care about its integrity, use a text editor like EDT on the VMS system to inspect it and make any necessary repairs.

Here is a more complicated example in which we upload a text file to an IBM mainframe through a direct (nondialup) linemode connection. The operating system is VM/CMS, which does not include a command like "cat" or COPY that will copy keystrokes to a file. In this case, a text editor, XEDIT, will be used.

Unfortunately, there is a small hitch here. XEDIT leaves text insertion mode if it receives a blank line. So if the file to be uploaded contains empty lines, Kermit must be told to insert a nonblank character into each blank line. The command for this is:

SET TRANSMIT FILL-EMPTY-LINE NONE *or* **SPACE** *or character*

This tells Kermit what to do when transmitting an empty line. The default action, NONE, means just send the empty line. SPACE means send a space character (this won't work with XEDIT, but it might work with other editors). Otherwise, you can include a nonblank character like X:

```
MS-Kermit>set transm fill X
```

Let's use this feature to transmit a familiar file to XEDIT on the IBM mainframe:

```
C>kermit                              (Start Kermit on the PC)
MS-Kermit>set speed 9600              (Set the speed)
MS-Kermit>set duplex half             (Connection is half duplex)
MS-Kermit>set flow none               (No Xon/Xoff flow control)
MS-Kermit>set handshake xon           (Must use handshake)
MS-Kermit>set parity mark             (Mainframe uses MARK parity)
MS-Kermit>connect                     (Begin terminal emulation)

VIRTUAL MACHINE/SYSTEM PRODUCT--CUVMB --PRESS BREAK KEY

Alt-B                                 (Hold down the Alt key and press B)
                                      (to send a BREAK signal)
!
.login sari                           (Log in)

Enter password: XXXXXXXX              (Half duplex; password echos)

LOGON AT 15:28:14 EDT THURSDAY 09/21/89
VM/SP REL 5 04/19/88 19:39
.
CMS
.xedit oofa txt                       (Start the editor)
.i                                    (Put it in text input mode)
DMSXMD573I Input mode:
Alt-X                                 (Hold down the Alt key and press)
                                      (the X key to return to PC Kermit)

MS-Kermit>set transm fill X           (Fill blank lines with X)
MS-Kermit>transmit oofa.txt           (Send the file)
MS-Kermit>connect                     (Return to the mainframe)
                                      (Press the Enter key)
                                      (to leave text input mode)
DMSXMD587I XEDIT:
.save                                 (Tell XEDIT to save file)
.qq                                   (Quit from XEDIT)
Ready; T=0.02/0.06 15:29:11
.lf oofa                              (Make sure file is there)
OOFA     TXT                          (It is)
Ready; T=0.01/0.01 15:29:16
.type oofa txt                        (Take a look at it)

This is the first line of the file oofa txt.
(This space intentionally left blank...)
And this is the last line.

Ready; T=0.01/0.01 15:29:20
.logout                               (Log out from the mainframe)
CONNECT= 00:01:11 VIRTCPU= 000:00.12 TOTCPU= 000:00.32
LOGOFF AT 15:29:25 EDT THURSDAY 09/21/89
Alt-X                                 (Hold down Alt and press X)
MS-Kermit>
```

Chapter 13

International Character Sets

The need for computers to communicate with each other—to share information—has become universal. But until recently, most computer hardware and software vendors provided their customers only with products based on the English language and character sets capable only of representing English text. These character sets contained the letters A–Z but none of the special characters required by other languages like French, German, Italian, Norwegian, Hebrew, Arabic, Greek, Russian, Chinese, or Japanese.

There are now computers capable of representing these special characters, but each may do so in a different way. For example, an importer of foods might create an order for pâté on an IBM PC, then transfer the order using Kermit (or any other file transfer protocol) to a supplier's Macintosh, where the word appears as "pÉtä" on the screen, which could easily result in a shipment of pita bread rather than goose liver.

A new extension to the Kermit protocol reduces this problem to manageable size by specifying a common *transfer syntax* consisting of a small number of well-defined standard character codes. During file transfer, each Kermit program converts between its own computer's character sets and the standard ones and needs no knowledge of any other computer's character sets.

MS-DOS Kermit 3.0 is the first Kermit program to follow the new protocol. MS-DOS Kermit's international character support includes not only file transfer but also terminal emulation.

IBM PC Character Sets

Characters are represented in the computer as fixed-size strings of bits, usually 7 or 8 bits per character. When such a bit string is transmitted to a display device like a terminal or printer, it is interpreted as a number that tells the position in the device's character generator. For example, the bit pattern 01000001 is equivalent to the decimal number 65, which in ASCII denotes the letter A (see Table II-4). When a video terminal receives this code, it retrieves the 65th element from its character table, which contains the dot pattern that looks like an A when displayed on the screen.

These are arbitrary associations in that any character (letter, number, symbol, and so on) could have been assigned any bit string. These associations are referred to as "character sets." If every computer used a different character set, no two computers could communicate. For example, an A on one computer might be interpreted as a B on another, and that wouldn't be very useful. For this reason, standard character sets like ASCII were developed. Entire books are dedicated to the topic of character set design.[15] But since ASCII cannot represent all the characters in every language, other character sets must also exist. And depending on your needs, you might have to use more than one of them.

IBM's character sets for the PC are called *code pages*. They are listed at the back of your DOS manual. Each of these character sets is a "superset" of ASCII that includes 128 additional printable characters, plus special graphic symbols like smiley faces, suits of cards (♣ ♦ ♥ ♠), musical notes, and arrows where the control characters usually reside. In true ASCII, the control characters do not have associated graphic symbols.

But the placement of the other 128 characters differs. The original PC code page, CP437, includes about 56 "national characters" that are not defined in ASCII, such as umlaut-u (ü), a-acute (á), c-cedilla (ç), and o-circumflex (ô); sufficient, according to the DOS manual, to support five different languages.

To capture markets in lands where the Five Languages are not spoken, IBM began to produce PCs with code pages for additional languages: a Portuguese code page, a Norwegian code page, and so forth. Eventually IBM added the multilingual (many-tongued) code page, CP850, which claims to support eleven languages and to make the earlier code pages obsolete.

Accented characters, Greek letters, and so forth, are referred to as *special characters*, even though there is nothing special about them to those who use them every day in their writ-

---

[15]See, for example, *Coded Character Sets, History and Development* by C.E. Mackenzie, Addison-Wesley (1980).

ten languages. They are special only because they are not part of ASCII and because each computer manufacturer seems to have a different way of entering and representing them.

First, let's find out how to enter international text on a PC—how to type it at the keyboard and how it is displayed on the screen. The IBM PC-PS/2 family has two ways to do this, the new way and the old way. The new way is more flexible and convenient, but it can't be done on all PCs.

Code Page and Keyboard Switching

This is the new way. The features described in this section are available only on PCs or PS/2s with DOS 3.30 or later that have an Enhanced Graphics Adapter (EGA) or EGA-compatible display board. Skip this section if this does not apply to you.

Old code pages never die. They don't even fade away. CP850 was able to add new western European national characters only by removing the Greek and technical characters from the other code pages. But since many users have applications that depend on the characters that were sacrificed, IBM equipped DOS 3.30 with a new feature called *code page switching*. This lets files created with different code pages coexist on the same PC. To work with files created with a given code page, you must make that your *active* code page, using the DOS CHCP command, for example:

```
C>chcp 865
```

You can find out what your current code page is by typing `chcp` by itself:

```
C>chcp
Active code page:  865
```

On most PCs, there are no keys labeled with special characters. How do you type an umlaut-u or an A-ring when there are no keys for these characters?[16] In DOS 3.30 or later, you may use the DOS KEYB command to reload your keyboard with the characters of a given national keyboard. Of course, this doesn't change the labels on your keycaps, so you have to refer to the appropriate keyboard template at the back of your DOS manual (Appendix E for DOS 3.30) to see which key to press for which character.

But before you can change code pages and keyboards, you have to make entries like these in your `AUTOEXEC.BAT` file:

```
REM ... Keyboard and console code page
NLSFUNC C:\COUNTRY.SYS
MODE CON: CP PREPARE=((850,,437) C:\EGA.CPI)
```

---

[16]PCs may be equipped with "national keyboards," which do indeed have keys for the special characters of a particular language. But the problem remains. There still must be a way to type, say, Italian characters on a Norwegian keyboard.

```
MODE CON: CP SELECT=850
KEYB US,850,C:\KEYBOARD.SYS
```

This sets you up with the United States keyboard layout and the multilingual code page on an EGA system. If you need characters from CP437 (like the Greek letters), you can switch to that code page:

```
C>chcp 437
```

and then back again to CP850 as often as you like. You can also change your keyboard at any time during your DOS session:

```
C>keyb it,850                    (Italian keyboard, CP850)
C>keyb                           (Check it)
Current keyboard code: IT  code page: 850
Current CON code page: 850
```

The first operand of the KEYB command is the country code, IT for Italy, NO for Norway, and so on. The DOS manual contains a table of the permissible country codes and which code pages may be used with them.

Most national keyboard configurations also allow special characters to be entered using two-character sequences. For example, the Norwegian configuration lets you type a wide variety of accented vowels that don't actually appear on the Norwegian keyboard. You press *both* the accent grave key and the desired vowel. For example, to produce A-grave, first press the accent grave key and then press the A key. IBM refers to these key sequences as dead-key combinations, and they are listed in the keyboard section of the DOS manual. Dead keys are not supported for U.S., U.K., or Italian keyboards.

Special Character Entry with Alt-Key Combinations

This is the old way of entering special characters, and it is available in any version of DOS or any PC model. On PCs without code page or keyboard switching, it is also the only way to enter special characters. This method requires no special entries in your AUTOEXEC.BAT file.

To enter a special character, hold down the Alt key, press three digits on the numeric keypad, and then let go of the Alt key. The three digits are the *decimal* representation of the IBM PC character in the current code page. For example, if you hold down the Alt key and press the numbers 1 2 8 consecutively using any code page, C-cedilla will be the resulting character on your screen. Unfortunately, the DOS manual code-page tables do not list the decimal values, only the *hexadecimal* values, which makes this method very difficult. Instead, you can use Table II-5, which lists the decimal codes as well as the hexadecimal codes for the special characters in all five IBM PC code pages.

Terminal Emulation

MS-DOS Kermit is fully capable of displaying and transmitting special characters during terminal emulation, provided it has these characters in its current code page and you have told Kermit to emulate the VT320 terminal (the only terminal emulated by MS-DOS Kermit that supports international characters):

```
MS-Kermit>set terminal vt320
```

Host computers such as UNIX systems, VAX/VMS, and IBM mainframes use different codes from the PC to represent national characters. Therefore, during terminal emulation, MS-DOS Kermit must translate the characters sent by the host computer into the corresponding PC symbols for display on the screen, and it must translate the characters you type on the PC keyboard into the host computer's character set (see Figure 13-1).

To perform the correct translations, MS-DOS Kermit must be told which character set the host computer is using. The command is:

SET TERMINAL CHARACTER-SET *name*

where you can pick any *name* from Table 13-1, for example:

```
MS-Kermit>set term char german
```

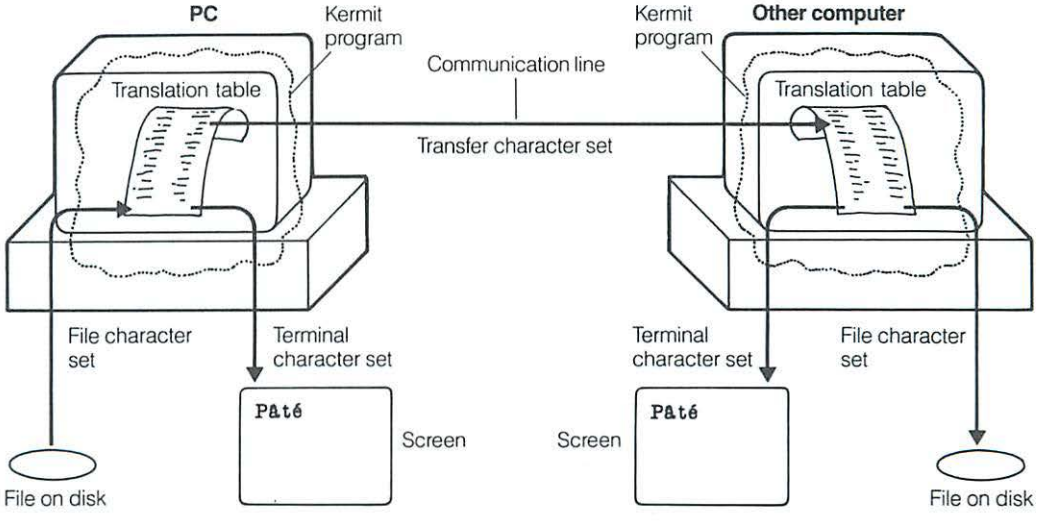


Figure 13-1 International Character Set Translation

Table 13-1 MS-DOS Kermit Terminal Character Sets

Name	*Bits*	*Characters*	*Description*
ASCII	7	128	United States ASCII
BRITISH	7	128	United Kingdom ASCII
DUTCH	7	128	NRC for the Netherlands
FINNISH	7	128	Finnish NRC
FRENCH	7	128	French NRC
FR-CANADIAN	7	128	NRC for French Canada
GERMAN	7	128	German NRC
ITALIAN	7	128	Italian NRC
NORWEGIAN/DANISH	7	128	NRC for Norway and Denmark
PORTUGUESE	7	128	Portuguese NRC
SPANISH	7	128	Spanish NRC
SWEDISH	7	128	Swedish NRC
SWISS	7	128	NRC for Switzerland
DEC-MCS	8	256	DEC Multinational Character Set
LATIN-1	8	256	ISO Latin Alphabet Number 1
TRANSPARENT	8	256	Current IBM PC Code Page

Table 13-1 shows the character sets supported by the DEC VT320 terminal that MS-DOS Kermit is emulating. The default terminal character set is Latin-1.

Kermit's built-in terminal character sets include all those supported by the international version of the VT320. These fall into two categories: 7-bit character sets (128 characters each) and 8-bit character sets (256 characters each). The 7-bit sets include US ASCII and country-specific variations of ASCII called National Replacement Character (NRC) sets. In an NRC set, certain nonalphabetic ASCII characters like `@`, `[`, `\`, `]`, `{`, `|`, `}`, and `~` are replaced by the characters needed for each language, as shown in Table 13-2. The NRCs are for use in 7-bit transmission environments, when parity prevents the use of the full 8 data bits. The terminology in the table is:

Acute An acute accent; a-acute is á

Grave A grave accent; a-grave is à

Ring A small ring over the letter; a-ring is å

Slash A slash through a letter; O-slash is Ø

Table 13-2 Selected National Replacement Character Sets

Code	*ASCII*	*German*	*Norwegian*	*French*
64	@ at sign	section	at sign	a-grave
91	[left bracket	A-umlaut	AE-digraph	degree sign
91	\ backslash	O-umlaut	O-slash	c-cedilla
93	] right bracket	U-umlaut	A-ring	section sign
96	` accent grave	accent grave	accent grave	accent grave
123	{ left brace	a-umlaut	ae-digraph	e-acute
124	| vertical bar	o-umlaut	o-ring	u-grave
125	} right brace	u-umlaut	a-ring	e-grave
126	~ tilde	ess-zet	tilde	umlaut

Umlaut Two dots over the letter; u-umlaut is ü

Cedilla A hook under a letter; c-cedilla is ç

Tilde A tilde over the letter; n-tilde is ñ

The 8-bit terminal character sets include the international standard ISO[17] Latin Alphabet Number 1, plus several DEC-specific sets: the Multinational Character Set (very similar to Latin-1), a Supplemental Graphics Set, a Special Graphics Set, and several others. In most international environments that allow 8-bit character transmission (no parity), the Latin-1 set is used. The default terminal character set is LATIN-1.

According to ISO Standard 8859-1, Latin Alphabet Number 1 supports at least the following languages: Danish, Dutch, English, Faeroese, Finnish, French, German, Icelandic, Irish, Italian, Norwegian, Portuguese, Spanish, and Swedish (to which we can add Latin itself for a total of fifteen supported languages).

Before you can use an 8-bit terminal character set, you must issue the MS-DOS Kermit commands SET PARITY NONE and SET DISPLAY 8. Refer to Table 13-1 to determine which character sets are 7 bits and which are 8 bits.

[17]International Organization for Standardization.

Controlling the Screen Display

MS-DOS Kermit's built-in terminal character sets are standard and should meet the needs of those who must display characters in any of the languages listed. You may, however, override Kermit's translation of incoming characters on an individual basis using the SET TRANSLATION INPUT command. The format is:

SET TRANSLATION INPUT *input-character screen-character*

which causes Kermit to translate the specified incoming character to the specified screen character. You may issue as many of these commands as you like. This will let you adapt MS-DOS Kermit to virtually any host character set, for example:

```
MS-Kermit>set transl in \123 \132   ({ to umlaut-a)
MS-Kermit>set transl in \125 \129   (} to umlaut-u)
MS-Kermit>set transl in on          (Turn it on)
```

This is called a user-defined translation, and it takes precedence over the currently selected terminal character set for the affected characters only. You must turn your user-defined translations on as shown in the example before they will take effect, and you can turn them off with the command SET TRANSLATION INPUT OFF, which will restore the normal translations of the current terminal character set.

Kermit's normal direction of screen display during connect mode is left to right. You can reverse this for languages like Hebrew and Arabic using the command SET TERMINAL DIRECTION RIGHT-TO-LEFT. This command will work for other languages (like English) too:

```
MS-Kermit>set term dir right
MS-Kermit>connect
                                                    gmc :nigoL
                                                     :drowssaP
                                          0.5 SMV/XAV ot emocleW
                                                              $
```

Leonardo da Vinci would have liked this command.

Keyboard Translations

Besides governing how incoming characters are translated for display on the screen, the SET TERMINAL CHARACTER-SET command also causes Kermit to translate any special characters that you type at the keyboard into the host character set you have selected. Recall that you can type special characters using any of the methods described above:

Alt-digit-digit-digit, dead key, specially labeled keys on national keyboards, or by loading a soft[18] national keyboard with the DOS KEYB command.

Just as you can override the translation of characters that arrive *from* the host with the SET TRANSLATE INPUT command, you can also override the translations of your keystrokes before they are sent *to* the host on an individual basis using the SET KEY command. For example:

```
MS-Kermit>set key \2334 \91
```

will cause *Alt-a* to send the German NRC code for a-umlaut. You can use SHOW KEY to find out scan codes (or consult Table II-6), and the NRC a-umlaut value came from Table 13-2. This use of SET KEY requires you to have explicit knowledge of the host character set codes for special characters.

Taking Advantage of Kermit's Terminal Character Sets

If you are a European computer user, the benefits of Kermit's multinational terminal emulation are obvious. You have probably been using a "national" character set for years, and now Kermit gives you a more convenient way to do this on your PC—you can type your ñ key, the correct code is automatically sent to the host, and the host's echo is automatically displayed as ñ on your screen.

In the English-speaking world, the benefits are not so obvious. But consider this: Except within certain very restricted environments, we have never been able to:

- Spell our overseas customers' name and addresses correctly.
- Correspond with an overseas client in his or her own language (even if we know it).
- Include foreign words and phrases in our documents.

Granted, this kind of thing was previously possible if we restricted ourselves to only one manufacturer's PCs and printers. Now we can use our PCs in conjunction with diverse mainframes, minicomputers, and other kinds of PCs that use different character sets, and Kermit will do all the necessary translation.

[18]Soft is computer slang for an effect accomplished by software.

When the Host Computer Has No Special Characters

You can even use MS-DOS Kermit to teach an English-only host computer a new language. Here's one possible way to do it for traditional UNIX.[19] Simply SET TERMINAL CHARACTER-SET to the NRC set for the desired language, say, German. Enter the special German characters (ä, ö, and ü, and β) using the German keyboard layout selected by the DOS command:

```
KEYB GR, 850
```

or using the Alt-key combinations shown in Table II-5. This setup allows all UNIX host applications to display your German characters correctly, for example:

```
C>chcp 850
C>keyb gr,850
C>kermit
MS-Kermit>set term vt320
MS-Kermit>set term char german
MS-Kermit>connect
Grüße aus UNIX!
$ help
UNIX hilft dem, der sich selbst hilft!
$
```

For a demonstration of Kermit's terminal character set support on the PC, REPLAY the file CHARDEMO.VT from the Kermit distribution diskette:

```
C>kermit
MS-Kermit>set terminal vt320
MS-Kermit>replay chardemo.vt
```

Do this several times, changing your terminal character set each time (for example, SET TERM CHAR FRENCH), and watch how columns are displayed differently. Different characters are shown in certain columns, depending on Kermit's current terminal character set.

Printer Control

The international characters that appear on your screen during terminal emulation can also be printed correctly on a printer attached to your PC if the printer supports your current IBM code page. Recent models of IBM printers also support code page switching via CHCP (see your DOS or printer manual for details).

[19]UNIX has always been a 7-bit ASCII environment, so 8-bit international character sets could not be used. The current trend is for each vendor to modify UNIX to be "8-bit clean," but it may take a long while for the results of this effort to become generally available.

Terminal Emulation Summary

Here is a brief recapitulation of the steps necessary to send and receive international characters to and from the host during terminal emulation:

1. Find out what your current code page is:

```
C>chcp
Active code page: 437
C>
```

2. Look up your current code page at the back of the DOS manual or the back of this book. Make sure it contains all the characters you need to use. If it does not, and if you are using DOS 3.30 or higher and have an EGA-compatible display board, install the code page you need by using a text editor like EDLIN to add the following lines to your AUTOEXEC.BAT file:

```
NLSFUNC C:\COUNTRY.SYS
MODE CON: CP PREPARE=((850,,437) C:\EGA.CPI)
MODE CON: CP SELECT=850
KEYB US,850,C:\KEYBOARD.SYS
```

   This example assumes that the DOS files COUNTRY.SYS, KEYBOARD.SYS, and EGA.CPI are in the top-level directory of your C disk. The MODE and KEYB commands add code page 850 (which should be the only one you need besides 437) and make CP850 your startup code page. In the KEYB command, you may substitute a different country code for US.

3. Reboot your PC: Press the Ctrl, Alt, and Del keys all at the same time.

4. Type the DOS KEYB and CHCP commands to check that the new AUTOEXEC.BAT commands took effect:

```
C>chcp
Active code page:  850
C>keyb
Current keyboard code: US Code page: 850
Current CON code page: 850
```

5. Run MS-DOS Kermit:

```
C>kermit
```

6. Issue a SET TERMINAL CHARACTER-SET command to tell Kermit which character set your host is using, and then issue the CONNECT command:

```
MS-Kermit>set parity none
MS-Kermit>set display 8
MS-Kermit>set term char latin1
MS-Kermit>connect
```

7. Enter special characters using Alt-key combinations (Alt-digit-digit-digit, see Table II-5), or specially labeled keys on a national keyboard (keys are labeled properly), or by loading a soft keyboard with the DOS KEYB command and referring to the appropriate keyboard template at the back of the DOS manual.

8. Using a text editor on the host, create a file with a few of these characters and then save it. Next, display the host file (for example, using the host TYPE command), and check that the special characters are displayed correctly.

9. If desired, customize your display and keys using Kermit's SET TRANSLATE INPUT and SET KEY commands.

File Transfer

MS-DOS Kermit can adapt itself to different host character sets during file transfer too, but only if it knows exactly what character sets are involved. MS-DOS does not record which file was created with which code page. This is something you have to remember for yourself so that you can tell Kermit how to translate it. If you use more than one code page, you might want to adopt some convention for naming your files, for example:

```
437JOHN.TXT
850VACE.TXT
```

or for storing them in separate directories.

Character set translation is done only for text files. If you give the command SET FILE TYPE BINARY, no translation will be done. Kermit uses SET FILE TYPE TEXT by default.

International character sets are a very recent addition to the Kermit protocol. You can use this feature only when the other Kermit also supports it. As of this writing, the following Kermit programs were under development to add these features:

- C-Kermit 5A (for UNIX, VAX/VMS, and OS/2).
- Macintosh Kermit 1.0 or later.
- IBM 370 Mainframe Kermit 4.2 or later (for MVS/TSO and VM/CMS).

Basic Principles of International Text File Transfer

1. The Kermit program that is sending the file must be told the character set that the file is written in. The command is SET FILE CHARACTER-SET. It is required because, in general, the sending computer has no way of finding this out for itself. How many computers have you seen where the DIRECTORY command lists the file's character set?

An IBM PC text file cannot contain a mixture of code pages; each file uses exactly one code page.[20]

2. The Kermit program that is receiving the file must be told what character set to use when writing the new file to disk. The command here is also SET FILE CHARACTER-SET. This command is necessary when there is a choice of character sets on the receiving computer.

3. *Both* Kermit programs must be told what character set to use for transferring the file. The command is SET TRANSFER CHARACTER-SET. Since there are hundreds of computer manufacturers in the world, and no one to control what codes they use to represent characters, it is clearly impractical to require that Kermit have knowledge of *every* computer's codes. That's why a standard, intermediate transfer character set is used.

Once the three character sets have been specified, the sending Kermit reads characters from its local disk, translates them into the transfer character set, puts them in Kermit packets, and sends them to the receiving Kermit, which in turn translates these characters into its own local file character set and writes them to a disk file:

PC file character set ↔ Transfer character set ↔ Host file character set

```
MS-Kermit>set file char cp850          (PC Kermit...)
MS-Kermit>set transf char latin1
MS-Kermit>connect
C-Kermit>set file char dec             (Host Kermit...)
C-Kermit>set transf char latin1
```

International Text File Transfer Commands and Options

The basic commands for international text file transfer are SET FILE CHARACTER-SET and SET TRANSFER CHARACTER-SET.

SET FILE CHARACTER-SET *name*

Tell MS-DOS Kermit the file's code page (see Table II-5), one of the following:

CP437 The original IBM PC code page

CP850 The multilingual code page

[20]Multilanguage documents are composed on the PC using proprietary word processing programs that have private encodings that are, in general, not compatible with any other DOS application. For purposes of file transfer, you should treat such files as *binary* files.

CP860 Code page for Portugal

CP863 Code page for French Canada

CP865 Code page for Norway

The default file character set is your current code page, as shown by the DOS command CHCP.

SET TRANSFER CHARACTER-SET *name*

Tell MS-DOS Kermit which character set to use when communicating with the other Kermit program:

LATIN1

ISO Standard 8859 Latin Alphabet Number 1 (see Table II-5). This is the default transfer character set, and it is capable of representing any alphabetic character with any diacritical mark in any of the IBM code pages listed under the SET FILE CHARACTER-SET command, except the dotless i and the Greek letters.

ASCII

Only 7-bit ASCII characters (see Table II-4) are used during file transfer. Special characters are translated to their closest ASCII equivalents, for example e-grave becomes simply e.

TRANSPARENT

Do not translate the IBM PC characters. Use this only when you want to keep the actual IBM PC codes intact when sending files to a different kind of computer and still retain Kermit's text file record format conversion. For compatibility with previous MS-DOS Kermit releases, TRANSPARENT is the default transfer character set.

The secret of successful international file transfer is knowing what character set to use on the remote host. You should use a character set supported by your host, the host version of Kermit, and the applications on the host that will be using the file. Some hosts support 8-bit character sets like ISO Latin-1, others support their own proprietary international character sets (like IBM mainframe Country Extended Code Pages), and others support only 7-bit codes like the ASCII and NRC sets. To find out what character sets your host Kermit supports: (a) read the documentation, (b) type `set file character-set ?` and `set transfer character-set ?`, or (c) use the HELP SET FILE CHARACTER-SET and HELP SET TRANSFER CHARACTER-SET commands, if available.

Examples of International Text File Transfer

The following examples show only the actual file transfer. By now, you should have grasped the concept that in order to transfer files, you have to set the appropriate parameters, connect to the other computer, and then log in; when you're finished using the other computer, you should connect back and then log out.

PC to VAX/VMS with Latin-1

PC code page 437 ↔ Latin-1 ↔ Latin-1

The document we're transferring is written in French on the PC using code page 437. The translation is CP437 to Latin-1. The transfer character set, Latin-1, is one that VAX/VMS computers know and support. We tell both Kermits which local file character set to use and which transfer character set is being used between them.

```
MS-Kermit>set file character-set cp437
MS-Kermit>set transfer character-set latin1
MS-Kermit>connect                (Connect to the VAX computer)
$ kermit                         (Start Kermit on VAX computer)

C-Kermit>set file character-set latin1
C-Kermit>set transfer character-set latin1
C-Kermit>receive
Alt-X                            (Escape back to the PC)
MS-Kermit>send french.txt        (Send French text)

  (The file is transferred...)

MS-Kermit>set terminal vt320     (Must use VT320)
MS-Kermit>set display 8          (To view 8-bit characters)
MS-Kermit>set parity none        (Ditto)
MS-Kermit>set term char lat      (Note abbreviation)
MS-Kermit>connect                (Connect to the host)
C-Kermit>exit                    (Exit from host computer's Kermit)
$ type french.txt                (Look at the result)

Un serveur Kermit n'est qu'un programme Kermit qui
fonctionne de manière spéciale.  Son
comportement est très similaire à
celui du Kermit ordinaire lorsque vous lui
adressez la command RECEIVE.  Il attend la
réception d'un message de la part de
l'autre Kermit, mais dans le cas du Kermit serveur,
le message est une commande disant ce qu'il faut
faire, normalement envoyer ou recevoir un fichier
ou un groupe de fichiers.

$
```

To work with the file on VAX/VMS, your terminal type must be VT320, and the terminal character set must be the same as the character set in which the file was stored on the VAX, namely, Latin-1.

PC to IBM Mainframe with CECP37

PC code page 850 ↔ Latin-1 ↔ CECP37

Here we transfer a document in French, German, Italian, Norwegian, Icelandic, etc., or any combination of these, written on the PC using code page 850, to an IBM mainframe that uses Country Extended Code Page (CECP) 37, which is an entirely different code that contains the same special characters as CP850 but in different positions.

```
MS-Kermit>set file character-set cp850
Kermit-TSO>set transfer character-set latin1
MS-Kermit>connect                  (Connect to the mainframe)
. kermit                           (Start Kermit on mainframe)
Kermit-TSO>set file character-set cecp37
Kermit-TSO>set transfer character-set latin1
Kermit-TSO>receive
Alt-X                              (Escape back to the PC)
MS-Kermit>send saga.txt            (Send an Icelandic saga)
```

Although Kermit lets us transfer international text files between IBM mainframes and PCs, the mechanism used to view the file once it is on the mainframe is highly dependent on the host hardware and in some cases may not be possible at all. Check your local IBM mainframe documentation.

PC to UNIX with Only ASCII

PC code page 850 ↔ Latin-1 ↔ 7-bit ASCII

Here we have a file written in German on the PC, using either code page 437 or 850. (For German, it doesn't matter which one is used since both of them include all the characters needed for German.)

```
Um interaktiv mit dem Programm zu arbeiten, ruft
man es von der DOS-Kommando-Ebene auf, indem man
den Programmnamen eingibt, normalerweise "kermit".
Wenn der Eingabe-Prompt "MS-Kermit>" erscheint,
können nacheinander beliebig viele Kermit-Befehle
eingegeben werden, bis man fertig ist und das
Programm verlaßen möchte. Die Befehle EXIT oder
QUIT bringen einen zurück zu DOS.
```

We want to transfer this file to a UNIX system, which is strictly 7-bit ASCII and has no way of storing the accented German characters. Can this be done without loss of information? Luckily, there is a rule for removing umlauts from German: Each vowel that has an umlaut can be replaced by the same vowel followed by the letter e, for example ä becomes ae. The other German special character, β, can be replaced by ss. C-Kermit will perform these translations if you tell it the language is German:

```
MS-Kermit>set file character-set cp850
MS-Kermit>set transfer character-set latin1
MS-Kermit>connect                  (Connect to UNIX)
% kermit                          (Start Kermit there)
C-Kermit>set language german       (Specify the language)
C-Kermit>set file character-set ascii
C-Kermit>set transfer character-set latin1
C-Kermit>receive
Alt-X                              (Escape back to PC)
MS-Kermit>send deutsch.txt         (Send the German file)
MS-Kermit>connect                  (Look at the results)
C-Kermit>exit
% cat deutsch.txt

Um interaktiv mit dem Programm zu arbeiten, ruft
man es von der DOS-Kommando-Ebene auf, indem man
den Programmnamen eingibt, normalerweise "kermit".
Wenn der Eingabe-Prompt "MS-Kermit>" erscheint,
koennen nacheinander beliebig viele Kermit-Befehle
eingegeben werden, bis man fertig ist und das
Programm verlassen moechte. Die Befehle EXIT oder
QUIT bringen einen zurueck zu DOS.
```

Notice the change in the words können, möchte, verlaβen, and zurück.

Kermit may also do the same kind of conversion for other languages that have similar rules. For example, in certain Scandinavian languages, å can be written as aa. For languages that do not have these rules, Kermit simply strips the diacritical marks and leaves the letters bare. For example, French pâté becomes simply pate.

PC to PC

There are two methods of PC-to-PC file transfer, depending on whether the two PCs are using the same code page.

PC code page 850 ↔ PC code page 850

If both PCs are using the same code page (a common case), all translations can be skipped. This brings an added benefit: Any mixture of text and binary files can be transferred together in a single operation:

```
MS-Kermit>set file type binary
MS-Kermit>cd \dan
MS-Kermit>send *.*
```

If you are running MS-DOS Kermit 3.0 or later on the receiving PC, you will not have to give a SET FILE TYPE BINARY command to it;[21] the sending Kermit will tell the receiving Kermit automatically.

PC code page 865 ↔ Latin-1 ↔ PC code page 860

Suppose you are a Norwegian in Oslo who uses a PC with IBM's Norwegian keyboard and code page 865, and your company has a branch office in Lisbon, Portugal, that uses PCs that have Portuguese keyboards and CP860, and you want to send a report that you have written in Norwegian to your friend who is stationed at the Lisbon branch. Here we have two IBM PCs that use different character sets, and (as you can see from inspecting Table II-5) the two character sets do not have the same characters in them: The Norwegian characters are missing from the Portuguese set and vice versa.

Without Kermit's translations, your Norwegian O-slash would arrive as U-grave, and your other special characters would be changed in more subtle ways. When Kermit must translate between two Latin-based character sets that do not have equivalent characters, it makes its best attempt at producing a sensible result. In this case, it will remove the accents from Norwegian letters that don't appear with the same accents in Portuguese. So O-slash becomes simply O (Øre becomes Ore), A-ring becomes A, and so forth.

Shortcuts

Readers who live in countries where English is not the native language might feel that all this is a lot of effort to transfer files in their own native language. This effort can be reduced or entirely eliminated by using Kermit's defaults and other shortcuts:

- Type the MS-DOS Kermit command SHOW FILE to find out what your default file character set is. If it is the one you intend, then the SET FILE CHARACTER-SET command is unnecessary.
- When MS-DOS Kermit sends a file to another Kermit program, it notifies the other Kermit of the transfer character set *if* the other computer supports a feature called

[21] In earlier versions of MS-DOS, this command is not available and does not need to be specified in this particular example.

Chapter 14

Macros, Command Files, and Scripts

In this chapter, you learn about the customizations that can perfectly suit MS-DOS Kermit to your communications environment, no matter how complex, and about the shortcuts you can take to invoke any collection of settings or actions automatically when Kermit starts up, by typing a single command, or even by pressing a single key. All the settings and options described in the preceding chapters can be collected and condensed into files, commands, or keys that are so concise and easy to use that, once you have set them up, you can happily forget nearly all that you have learned so far.

MS-DOS Kermit is a powerful and flexible communication program. Because it gives you control over virtually every aspect of its operation, it can be adapted to nearly any situation. That's why there are so many commands! But its wealth of commands can make Kermit intimidating to the casual user and cumbersome even for the most experienced. Continue reading and you will learn how to make Kermit much easier to use, both for yourself and others. You will even be able to set up complicated "canned" procedures that the computerphobes in your office won't be afraid to use.

Command Macros

In Chapter 7, we looked at communication parameters like speed, parity, duplex, flow control, and handshake. If you must switch your PC frequently between two computers that are very different from each other—say, an IBM mainframe and a VAX/VMS system—would you want to type five or ten long SET commands each time you switched from one to the other? Of course you wouldn't.

Suppose you've read all the previous chapters and learned how to adapt your connection to the different types of host computers. Each time you used the IBM mainframe (in linemode), you would issue all these SET commands:

```
MS-Kermit>set flow none          (Communication settings)
MS-Kermit>set parity mark
MS-Kermit>set handsh xon
MS-Kermit>set duplex half
MS-Kermit>set speed 4800
MS-Kermit>set block 3            (Protocol settings)
MS-Kermit>set window 1
MS-Kermit>set rec pack 1000
MS-Kermit>set key \270 \8        (Backspace key setting)
```

And to switch to the VMS system, you would issue these SET commands:

```
MS-Kermit>set parity none        (Communication settings)
MS-Kermit>set handshake none
MS-Kermit>set flow xon/xoff
MS-Kermit>set duplex full
MS-Kermit>set speed 9600
MS-Kermit>set rec pack 200       (Protocol settings)
MS-Kermit>set window 8
MS-Kermit>set block 1
MS-Kermit>set key \270 \127      (Backspace key setting)
```

To switch back to the IBM mainframe, you'd have to enter the first group again, and to switch back to the VAX, you'd have to enter the second group again. And so on. If you need further motivation to continue reading, do this ten times.

Defining Macros

You can make up short slang words to refer to collections of settings and other commands. These words and the Kermit commands you assign to them are called *macros*. Once defined, a macro is used just like any other Kermit command. For example, let's create one macro for our IBM mainframe settings and a second one for VAX/VMS:

```
define ibm set parity mark, set handshake xon, set flow none,-
 set duplex half, set speed 4800, set receive packet-length 1000,-
 set window 1, set block 3, set key \270 \8
define vms set parity none, set handshake none,set flow xon/xoff,-
 set duplex full, set speed 9600, set receive packet-length 250,-
 set window 8, set block 1, set key \270 \127
```

The command for creating macros is DEFINE. The first word after DEFINE is the macro name. You can pick any name you like. After the macro name comes its definition, which is a list of Kermit commands separated by commas. The definition can be up to 255 characters long. You can use any valid abbreviations for the Kermit commands you include in your definition, and this is a good space-saving technique:

```
def vms set pa n, set h n, set fl x,-
 set dup f, set sp 9, set rec pack 250,-
 set wi 8, set bl 1, set k \270 \127
```

This definition has 99 characters, versus 174 in the original unabridged version. Both do exactly the same thing. Notice the dash that is used to continue a macro definition onto the next line for easy reading.

You can see the definition of a macro by typing the SHOW MACRO command at the MS-Kermit> prompt:

```
MS-Kermit>show macro vms
 VMS = set pa n<cr>
  set h n<cr>
  set fl x<cr>
  set dup f<cr>
  set sp 9<cr>
  set rec pack 250<cr>
  set wi 8<cr>
  set bl 1<cr>
  set k \270 \127<cr>
```

<cr> means carriage return (the character produced when you press the Enter key). This is what Kermit translates the comma between the commands into. All commas in macro definitions get this treatment, so if you want to include an actual comma in the definition, use \44 (the ASCII value of comma). If you type SHOW MACRO alone, Kermit will display all your macros. If you type SHOW MACRO C, Kermit will show all the macros that begin with the letter C.

When you invoke a macro by typing its name at the MS-Kermit> prompt, all the Kermit commands in the definition are executed in order:

```
MS-Kermit>ibm
```

You can also give the command DO before the actual macro name if this verb helps you to remember that an action is taking place:

```
MS-Kermit>do ibm
```

In either case, you might not believe that these parameters were set because there is no

message verifying this. When in doubt, you can check the settings with the SHOW COMMUNICATIONS command:

```
MS-Kermit>show communications
  Duplex: half              Parity: mark
  Handshake used: ^Q        No flow control used
```

(^Q means Ctrl-Q, which is the character associated with Xon.)

Now switch to the VAX/VMS settings and verify them:

```
MS-Kermit>vms
MS-Kermit>show communications
  Duplex: full              Parity: none
  Handshake used: none      Flow control: Xon/Xoff
```

Similarly, the protocol settings can be checked by typing `show protocol`.

You can define macros composed of any Kermit commands. Any command that you can type at the `MS-Kermit>` prompt can be included in a macro.

Noisy Macro Example

Just as you can set up macros to switch between the different types of computers you communicate with, you can also set them up for different physical conditions. Suppose you make a lot of modem calls for transferring files. Sometimes you get a clean, noise-free connection. At other times, you get a very noisy connection, resulting in garbage characters on your screen:

```
Wel}}}} to VAp+=MS              (Welcome to VAX/VMS)
Us||na^e:                       (Username:)
```

Recall the scenario from Chapter 9 where you had a clean communications line at work and a noisy one at home. On a clean line, you could keep the packet length at 94, give up after 4 retries, and use the least amount of error checking. On a noisy line, you could reduce the packet size to 40 characters instead of 94, allow 20 retries instead of only 4, and apply Kermit's strongest error check on the packets.

```
def clean set rec pac 94, set ret 4, set bl 1
def noisy set rec pac 40, set ret 20, set bl 3
```

Now when you make your connection, you can type the command CLEAN or NOISY depending on what your screen looks like.

Macros with Arguments

So far we've just used macros to combine a list of Kermit SET commands. Let's see what else macros can do.

Just like real Kermit commands, macros can take arguments. Can not! Can so! Can not! Can so! No, not that kind of argument. In computer jargon, an *argument* is like the object of a verb. The verb tells what is being done, and the object tells who or what it is being done *to*. In a macro, the macro name tells Kermit what to do, and the arguments tell Kermit how to do it, who to do it to, when to do it, how many times to do it, and so on.

For example, even though Kermit already has several DOS functions built into it (TYPE, DELETE, DIRECTORY), you can define macros that use Kermit's RUN command to include additional DOS commands so that you can stay in Kermit all day:

```
def more run more < \%1          (DOS MORE command)
def rename run ren \%1 \%2       (DOS REN command)
def copy run copy \%1 \%2        (DOS COPY command)
def edit run edlin \%1           (For editing files)
```

With these macros, you can issue DOS-like commands directly from the `MS-Kermit>` prompt:

```
MS-Kermit>more juri.txt
MS-Kermit>rename andrei.txt misha.txt
MS-Kermit>copy marina.c kostya.c
MS-Kermit>edit shamil.txt
```

The DOS MORE command is like TYPE, but it pauses after each screenful. It does not accept a filename directly, as TYPE does. The filename must be given using the DOS input redirection symbol, <. Kermit's MORE macro, as defined above, allows you to omit this symbol.

These macros illustrate the use of Kermit's positional parameters. In the RENAME macro:

```
RENAME                           (The macro name)
RUN REN \%1 \%2                  (The macro body)
\%1 \%2                          (The parameters)
```

`\%1` is the first parameter, `\%2` is the second, and so on, up to `\%9`. These parameters change into whatever filenames you type after the macro command RENAME.

```
MS-Kermit>rename boring.txt oofa.txt
```

`\%1` is expanded to `boring.txt`, and `\%2` is expanded to `oofa.txt`.

Later, we'll see how to put these parameters to other uses.

Macros on Keys

Now that you know how to define a macro and invoke it by typing a single word (possibly followed by arguments), you will be glad to learn that there is also a way to invoke a macro in a single keystroke.

There are two special macro names, TERMINALS and TERMINALR. What is special about them is that they have associated Kermit "keyboard verbs," \Kterminals and \Kterminalr. You can define these macros to do anything you like and then assign the associated verbs to the keys of your choice. You can then invoke any imaginable procedure with a single keystroke during terminal emulation:

```
define terminals set terminal direction left, connect
set key \315 \kterminals
define terminalr set term dir right, connect
set key \316 \kterminalr
```

This example sets up the F1 and F2 keys to change the direction of writing on the screen. This can be handy for entering Hebrew or Arabic words into an English or French document, or vice versa (assuming your host computer's word processor is set up for this capability).

Command Files

We've seen how to save typing by defining keys and macros, but typing the definitions is a lot of work in itself, and once you exit from the MS-DOS Kermit program, your macro definitions are forgotten. So we need some way to save macro definitions for future use. For this reason, Kermit is able to execute commands not only from the keyboard but also from disk files. If you give the command TAKE followed by a filename, Kermit will execute the commands that are in the file:

```
MS-Kermit>take defs.tak
```

This file could include all of your macro and key definitions. By convention, Kermit TAKE command files have the filetype .TAK.

Here is a handy use for a command file. Suppose you want to send many files to another computer, but since they are scattered all over the place, there is no way you can specify them in a single SEND command. But you don't want to sit at your PC for hours typing separate SEND commands for each. If you put the receiving Kermit in server mode, you can collect your SEND commands into a command file called, say, LUNCH.TAK:

```
cd \amy
send lavender.*
send \peter\dragons.txt
cd \joann
send *.*
send a:\kermit\mskermit.ini
bye
```

Now use this command file to send all of your files while you eat lunch:

```
MS-Kermit>connect                (Connect to the host computer)
$ kermit                         (Run Kermit on the host)
C-Kermit>server                  (Put it in server mode)
```

```
Alt-X                                   (Escape back to the PC)
MS-Kermit>take lunch.tak
```

You can create a command file using a text editor, like EDLIN, DOS's standard, no-frills line editor:

```
C>edlin defs.tak                        (Create file DEFS.TAK)
*i                                      (Insert text)
       *1:set key \315 I just typed F1.\13
       *2:set key \316 Did you type F2?\13
       *3:set key \317 F3 is my favorite.\13
       *4:Ctrl-C
*e                                      (Save file; exit EDLIN)
C>
```

To execute the file, run the MS-DOS Kermit program and give the TAKE command:

```
C>kermit
MS-Kermit>take defs.tak
MS-Kermit>connect
```

Now you're connected to the host, and your SET KEY commands take effect:

- Type the F1 key.
  (You should see the sentence:) `I just typed F1.`
- Type the F2 key.
  (You should see the sentence:) `Did you type F2?`
- Type the F3 key.
  (You should see the sentence:) `F3 is my favorite.`

Initialization Files

What if you forget to issue your TAKE command each time you run Kermit? If you want to have certain settings and definitions in effect all the time, you can collect them into a special file called `MSKERMIT.INI`, and they will be executed automatically every time you run Kermit. On other systems, Kermit initialization files have other names, like `CKERMIT.INI` for VAX/VMS C-Kermit or `.kermrc` for UNIX Kermit.

The `MSKERMIT.INI` file must be in your current directory or in your DOS PATH, as described in Chapters 2 and 3, or else Kermit won't find it. To reassure yourself that Kermit has executed your initialization file, it is a good idea to include an ECHO command in it, as shown below, which displays text on your screen. For example, the line:

```
echo Happy Birthday!\13
```

gives you the message `Happy Birthday!` when Kermit starts up.

Here's a sample MSKERMIT.INI file for an IBM PC that is connected to an IBM mainframe on port 1 and uses a modem on port 2 to dial up a VAX computer.

```
COMMENT - Set up modem parameters
COMMENT - for dialing up VAX systems...
  set port 2                          ; Select port 2
  set speed 1200                      ; Port 2 speed is 1200
  set parity none                     ; Port 2 parity is none
  set duplex full                     ; Port 2 duplex is full
  set flow xon/xoff                   ; Port 2 flow control
  define modem set port 2, set key \270 \127

COMMENT - Set up direct line
COMMENT - for IBM mainframe in linemode...
  set port 1                          ; Select port 1
  set speed 9600                      ; Port 1 speed is 9600
  set parity mark                     ; Port 1 parity is mark
  set duplex half                     ; Port 1 duplex is half
  set flow none                       ; Port 1 no flow control
  define ibm set port 1, set key \270 \8

COMMENT - General settings
  set warning on                      ; Protect my files
  set terminal cursor block           ; Use a big cursor
  set terminal screen reverse         ; Use reverse video
  echo Smile!
```

MS-DOS Kermit remembers port-related settings for each port, so all you have to do to switch between them is SET PORT 1, SET PORT 2. But since the IBM and VAX systems use a different code for backspacing, and this is not a port-related parameter, macros are defined to SET PORT and SET KEY for each system.

Command files may include COMMENT lines for documentation, and most Kermit commands in TAKE files can also have trailing comments that start with a semicolon (;), as shown above. Trailing comments should not be used with interactive commands or within macro definitions.

TAKE files may include TAKE commands. If you wish, you can have a TAKE file that TAKEs many other TAKE files or a TAKE file that TAKEs a TAKE file, which TAKEs another TAKE file, and so on to a depth of about 20.

MS-DOS Kermit command files can be as short and simple or as long and complicated as you want to make them. In fact, a major difference between a command file and a macro is that a command file's length is unlimited, whereas a macro can be no more than 256 characters long.

Script Programming

Gaining access to another computer can be a tedious and repetitive process: dialing your modem, waiting for the connection, typing your access codes, waiting for the system prompt—day after day the same thing. Aren't we supposed to be using computers to automate the tedious and repetitive processes for us?

Perhaps you're thinking that you could use Kermit's TRANSMIT command to send all the characters that you would normally type during this process out the serial port. Good idea, but it usually won't work. The reason it usually won't work is that you have to synchronize the characters you type with the prompts and responses of the computer or modem. Remember, things don't always go smoothly—sometimes you have to make decisions. For example, if your modem tells you NO ANSWER instead of CONNECT 2400, then you would (or should) *decide* not to type the login sequence.

If Kermit can trick the modems and computers into thinking that you are sitting there in person typing at them at each moment, then it is performing a kind of "human emulation." How can we coordinate Kermit's actions with the events that Kermit observes through the communication port?

As soon as we start talking about a computer making decisions, we're talking about *programming*. But don't let this word scare you if you're not a programmer. Kermit programs are called *scripts*. A script is just a list of Kermit commands, in either a command file or a macro or any combination of the two, that contains some special human emulation commands—mimicking what you would type, looking at the computer's response and deciding what to do, and trying things over again when they don't work. In other words, doing what you would do.

To automate a procedure, you must tell MS-DOS Kermit to issue all the commands that you would normally type when interacting with Kermit itself, the modem, and the remote computer. The one difference, which often leads to confusion, is that you must do all this without giving the CONNECT command. CONNECT reads text only from the keyboard, not from a command file or macro definition. A command file like:

```
set speed 9600
connect
atdt8765432
login peppi
```

would not work: The CONNECT command would be executed, and Kermit would wait for you to start typing. When you finally escaped back, Kermit would try (and fail) to execute the next two lines as Kermit commands. Only put a CONNECT command in a script when you really want to step in and take over the interaction yourself.

The INPUT and OUTPUT Commands

Two special commands, INPUT and OUTPUT, take the place of CONNECT for interacting with the modem or the remote computer:

INPUT

What you would expect to see on your screen. The INPUT command tells Kermit to look for the specified sequence of characters or to give up if this character sequence does not arrive after so many seconds. The format of the command is:

INPUT *n characters*

where *n* is the number of seconds to wait, and *characters* are what Kermit is to look for during those seconds. For example:

```
input 10 Username:
```

This command tells Kermit to wait up to 10 seconds for the VAX/VMS `Username:` prompt to arrive. If you leave out the number, Kermit will wait its default number of seconds (one, but you can change it):

```
input Username:
```

OUTPUT *characters*

What you would normally type. The OUTPUT command tells Kermit to send the given text out the communication port to the modem or the remote computer:

```
output I am not really typing this...
```

INPUT and OUTPUT text can contain any sequence of 7-bit or 8-bit characters. Most printable characters can be included literally, as shown above. Control characters or other nonprintable characters can be inserted by giving their numeric ASCII values, preceded by a backslash (see Tables 16-2 and II-4); for example:

```
output atdt7654321\13
```

sends a dialing command to a Hayes modem, followed by a carriage return (`\13`) (as if you pressed the Enter key). The OUTPUT command only sends the characters you specify, so be sure to include `\13` wherever you would press the Enter key. The OUTPUT string can also contain a special symbol, `\b`, to send a BREAK signal.

You can also include Kermit *variables* in these strings. For example:

```
define \%n 7654321
output atdt\%n\13
```

gives the same result as the previous example. The variable `\%n` is similar to the parameters `\%1` and `\%2` we saw in the macro examples, but unlike macro parameters, this variable remains permanently defined unless you undefine it or redefine it—whenever you refer to `\%n` from now on, Kermit will translate it to `7654321`. Permanent variables are

designated by \% followed immediately by a single letter. Uppercase and lowercase are equivalent: \%a is the same variable as \%A.

The behavior of the INPUT command can be controlled by a series of special SET commands:

SET INPUT CASE IGNORE *or* **OBSERVE**

Tells whether to pay attention to alphabetic case during INPUT. Case is ignored by default. This means that the command INPUT LOGIN would succeed if LOGIN or Login or lOgIn appears, for example:

```
set inp cas obs
```

SET INPUT DEFAULT-TIMEOUT *seconds*

Tells how long to wait for the INPUT string if the waiting time is left out of the INPUT command. For example:

```
set input default-timeout 10
input Username:
```

will cause the INPUT command to wait 10 seconds for the Username: prompt. The default timeout is one second if you don't change it with this command.

SET INPUT ECHO ON *or* **OFF**

Tells whether the characters that are read by the INPUT command are to be echoed on your screen. INPUT ECHO is ON by default, so you can watch. Turn it off for "silent running":

```
set inp echo off
```

Characters that are INPUT and then displayed on the screen are *not* passed through Kermit's terminal emulator, so screen-oriented host output may look garbled. This problem can be somewhat alleviated by using IBM's ANSI.SYS console driver.

SET INPUT TIMEOUT-ACTION PROCEED *or* **QUIT**

Tells what Kermit should do if an INPUT command fails to read the specified string within the given amount of time. PROCEED means just go on to the next command, and QUIT means exit from the current macro or command file immediately, without executing the remaining commands. The default setting is PROCEED.

In simple cases, INPUT and OUTPUT are the only special commands necessary for conducting an automated dialog. For example, if the following sequence was stored in a command file called VAX.SCR:

```
set speed 2400                  ; Communication settings
set input timeout quit          ; Give up if INPUTs fail
```

```
output ATDT5554321\13          ; Dial the Hayes modem
input 20 CONNECT               ; Wait 20 secs for CONNECT message
output \13                     ; Send a carriage return
input 10 Username:             ; Wait 10 secs for host prompt
output cmg\13                  ; Send my username
connect                        ; I'll do the rest myself
```

you could dial up and log in to your VAX simply by giving the command:

```
MS-Kermit>take vax.scr
```

And often it will actually work. When it doesn't, one of the INPUT commands will fail, and the `MS-Kermit>` prompt will return.

The PAUSE and CLEAR Commands

Why should this script fail? One reason is that it may take some time after the modem's `CONNECT` message for the communication channel to become completely clear. If the carriage return (`\13`) that is supposed to wake up the VAX is sent during this time, it could be lost. Another reason might be that the dialing command that was sent to the modem echoed back and confused matters. Here are some solutions:

PAUSE *seconds*

Does nothing for the specified number of seconds before executing the next command, for example:

```
pause 10
```

If the seconds are omitted, Kermit will pause for one second. The pause will be interrupted if you type anything on the keyboard.

CLEAR

Clears the serial port input buffer. If you know that characters have arrived from the host that are not important to the script, or that could confuse it, this command will discard them.

Here is the script with the PAUSE and CLEAR commands added:

```
set speed 2400                 ; Communication settings
set input timeout quit         ; Quit if any INPUT fails
output ATDT5554321\13          ; Dial the Hayes modem
clear                          ; Clear away ATDT echo
input 20 CONNECT               ; Wait 20 secs for CONNECT message
pause 2                        ; Give 2 secs for settling
output \13                     ; Send a carriage return
input 10 Username:             ; Wait 10 secs for host prompt
output cmg\13                  ; Send my username
connect                        ; I'll do the rest myself
```

The STOP, IF, and GOTO Commands

But there are many other possible reasons for failure:

- The modem was not turned on. No input will ever arrive in this case.
- The modem was not connected to the telephone, or the telephone was broken. The modem will respond `NO DIALTONE` instead of `CONNECT`.
- The modem call was not answered. The modem will respond `BUSY` or `NO ANSWER`.
- The VAX was down, and so the `Username:` prompt will not appear.

No matter how powerful Kermit is, it can't turn on your modem, fix your phone, or remedy other such problems. But it can give you informative messages so that you can correct the problems yourself. To do this, you'll need some more commands:

STOP

Exits from the script command file (or macro) immediately without executing any more commands and returns to the `MS-Kermit>` prompt.

IF SUCCESS *command*

Executes the following command only if the preceding INPUT command succeeded, for example:

```
if success echo It worked!
```

IF FAILURE *command*

Executes the following command if the preceding INPUT command failed, for example:

```
if failure echo It didn't work.
```

GOTO *label*

Instead of executing the next command, goes to the specified script label and begins executing commands there. A script label is on a line by itself, flush against the left margin, and begins with a colon (:). Here is an example of the use of GOTO and labels:

```
echo Skipping the next message
goto last
echo You should not see this!
:last
echo This is the last message
```

The label can be either before or after the GOTO statement. If it is before, you can create an infinite loop (don't try this at home folks!):

```
:loop
echo again and
goto loop
```

This will print "again and again and again and" forever. (You can interrupt this or any script program by typing `Ctrl-C`. You might need to do this several times.)

Let's apply these new commands to our example:

```
set speed 2400              ; Communication settings
set input timeout proceed ; Check failures with IF
output AT\13                ; See if modem is there
input 2 OK                  ; Should say "OK"
if success goto dial        ; If so, go and dial
echo Please connect or turn on your modem!\13
stop
:dial
output ATDT5554321\13       ; Dial the Hayes modem
input 20 CONNECT            ; Wait 20 seconds for CONNECT message
if success goto connected ; So far so good
echo Dialing failed - try again later.
stop
:connected
pause 2                     ; Settling time
output \13                  ; Send a carriage return
input 10 Username:          ; Wait 10 seconds for Username:
if success goto login
echo No VAX login prompt - try again later.
hangup
stop
:login                      ; Everything worked
output cmg\13               ; Send my username
connect                     ; I'll type the password myself!
```

Now we're getting a little more "user-friendly." This script should always work as intended: Either it will either succeed or it will tell you why it didn't.

The REINPUT, SET COUNT, and IF COUNT Commands

How much more intelligence can we add to our little procedure? A lot! Let's consider two more cases that we would handle routinely if we were making this connection manually:

- The modem says `BUSY` rather than `CONNECT`. Wait one minute and then try again. But patience runs out after doing this five times, so give up after that if there is no connection.
- Dialing is at 2400 bps, but the other modem answers at 1200 bps, so the local modem drops down to 1200 bps also and gives the message `CONNECT 1200`. But Kermit's speed is still set to 2400. You or I would notice this and adjust Kermit's speed, but so far our script program remains blissfully unaware that this has happened.

Here are the new commands we will need:

REINPUT *seconds characters*

Just like INPUT, except instead of reading characters as they arrive at the serial port, the REINPUT command rereads the ones that previously arrived. This way, messages from the modem or the host can be scanned several times for different words or phrases, like `CONNECT 2400` or `CONNECT 1200`.

SET COUNT *number*

Put a limit on the number of times a statement will be executed, for example:

```
set count 5
```

IF COUNT *command*

Subtract one from COUNT. If the result is greater than zero, Kermit executes the next command, for example:

```
if count goto loop
```

Here is an example of a counted (finite) loop:

```
set count 3
:again
echo Hello\13
if count goto again
echo Goodbye\13
```

This script says "Hello" three times and then says "Goodbye."

Before proceeding, you should be aware that overuse of IFs, GOTOs, and labels can make a program very unclear, hard to read, and therefore prone to programming errors as you modify it. Since our major use of these features is to issue a message and stop if there is an error, we can define a macro to do this and avoid labels and GOTOs in these cases:

```
define errstp echo Error: \%1\13,-
  define \%1,-
  hangup,-
  stop
```

This macro, named ERRSTP (short for ERROR STOP), echoes its argument, `\%1` (the first positional parameter, remember?), and then deletes it. The HANGUP command closes the telephone connection, and then the STOP command terminates the script and returns immediately to the `MS-Kermit>` prompt.

Positional parameters are assigned one word at a time. A statement like:

```
errstp Please turn on your modem.
```

would assign only the word "Please" to the parameter `\%1`, and so the error message would just be `Error: Please`. To allow more than one word to be assigned to a single

macro parameter, Kermit lets you group words within curly braces when using them after a macro name, for example:

```
errstp {Please turn on your modem.}
```

resulting in `Error: Please turn on your modem.`

By adding loops to our script, we can make it keep dialing until the other computer answers, up to whatever limit we give in the SET COUNT command. And by adding the REINPUT command, we can process the modem's messages more intelligently. While we're at it, we should remove the part about logging in to the VAX and connecting. This way the script will be an all-purpose Hayes modem dialer, which you can invoke from within any command file, script, or macro. To be truly general, however, the script needs one more modification: It must be able to call *any* phone number, not just 5554321. So in the Hayes ADTD dialing command, we will substitute the `\%1` parameter.

Let's call our new script file `HAYES.SCR`. It can be executed by typing `take hayes.scr` at the `MS-Kermit>` prompt. How can we pass the phone number as a parameter to this file? We could turn the command file into a macro, but it's much too long for that. Here's another way:

```
MS-Kermit>def \%1 555-1212
MS-Kermit>take hayes.scr
```

This is obviously not the sort of thing you want to type all the time. A better solution is to define a macro that TAKEs the command file:

```
define dial take hayes.scr
```

The DIAL macro automatically assigns its first argument, the phone number, to the first positional parameter, `\%1`.

```
MS-Kermit>dial 555-1212
```

Now, when the DIAL macro TAKEs the `HAYES.SCR` command file, `\%1` will contain the phone number so that the statement `output ATDT\%1\13` will call the given number.

```
def errstp echo Error: \%1\13,def \%1,hang,stop
set speed 2400                  ; Dial at 2400 bps
set input timeout proceed       ; Allow IF SUCCESS, IF FAILURE
set input echo off              ; Don't echo the modem test
output AT\13                    ; Send AT
input 2 OK                      ; Modem should say "OK"
if fail errstp {Turn on or connect your modem!}
clear                           ; Clear away old modem echoes
set input echo on               ; From now on, show what happens
set count 5                     ; Set up dialing loop
echo \13Dialing \%1, wait...\13\10
goto dial                       ; 1st time, skip Redialing message
:REDIAL
echo \13Redialing...\13\10      ; Message for redialing
```

```
:DIAL
output ATDT\%1\13                ; Dial the number
input 60 \10                     ; Wait for linefeed after message
if fail errstp {No response from modem.}
reinput 1 CONNECT                ; Got message, was it CONNECT?
if success goto speed            ; Yes, go check the speed
reinput 1 BUSY                   ; Not CONNECT, was it BUSY?
if failure errstp {No dialtone or no answer.}
Echo \13Busy...                  ; Phone was busy, give message
hangup                           ; Hang up
pause 60                         ; Wait one minute
if count goto redial             ; Then go redial

errstp {It's always busy!  I give up.} ; Too many tries.

:SPEED                           ; Connected!
reinput 1 1200                   ; Was message CONNECT 1200?
if success set speed 1200        ; Yes, change the speed
```

This script is just about as smart as you are! You can use it to type any DIAL command you like at the Kermit prompt:

```
MS-Kermit>dial 555-1212          (Local call)
MS-Kermit>dial 1-800-765-4321    (Long distance)
MS-Kermit>dial 3210              (Local extension)
```

Now that you've seen how to construct a powerful, intelligent, and robust DIAL command for a Hayes modem, you will be able to make DIAL commands for any other kind of modem, as well as for data PBXs, terminal servers, and any other kind of communication device that connects you to another system. And you can use the same techniques to construct scripts for logging in to any kind of computer or service.

A Dialing Directory

So that you don't have to remember the phone numbers for CompuServe, Telenet, Tymnet, and all the other computers and services that you call, you can construct a dialing directory that lets you refer to them by name by typing `dial telenet`, `dial tymnet`, and so on. Furthermore, if you type `dial` alone, it becomes a redial command; that is, it will call the previous number again (or if there was no previous call, it will complain).

As you probably have guessed, more new commands are needed:

ASSIGN *variable1 variable2*

The ASSIGN command copies the value of *variable2* into *variable1*. This differs from the DEFINE command, which copies the *name* of *variable2* into *variable1*. To demonstrate:

```
define \%a old
```

```
define \%b \%a
assign \%c \%a
define \%a new
echo \%a \%b \%c
```

The ECHO command will print `new new old`.

IF EQUAL *string string command*

This compares two character strings to see if they are the same (equal). If they are, the *command* is executed, for example:

```
if equal \%1 hello echo You typed hello!
```

IF = ARGC *number command*

The special Kermit variable ARGC tells the number of words in a macro invocation. For example, if you type `dial 1234`, then there are two. This form of the IF statement can be used to find out how many words followed the macro name, so that it can supply default values for fields the user left off, like the phone number in a DIAL command. The = symbol means "equal to." You can also use > for "greater than" and < for "less than," and you can prefix any of these with the word NOT to reverse their meaning, for example, "`NOT <`" means greater than or equal to. Remember: Use EQUAL for comparing character strings. Use =, <, and > for comparing numbers.

IF DEFINED *variable*

This statement succeeds if the given variable is defined and fails if the variable isn't defined, for example:

```
if defined \%n goto gotit
```

You can reverse the sense of this statement (like any IF statement) using NOT:

```
if not defined \%n goto needit
```

Let's use these features to construct our new DIAL macro, complete with dialing directory and recall of the last number dialed. First we use:

```
if = argc 2
```

to check whether the user supplied a phone number. If the user typed just `dial`, ARGC would be 1, but if the user typed `DIAL 5554321`, ARGC would be 2. For redialing, the trick is to store the previously dialed number in a permanent variable. We will use `\%n`:

```
if = argc 2 assign \%n \%1,-
if < argc 2 if not def \%n fatal {Dial what?},-
```

If the user types just `dial` and the variable `\%n` is not yet defined, then the DIAL macro says, `Dial what?` and quits (using the FATAL macro, which just prints its argument and stops). But if a number *was* previously dialed, the macro will find it in `\%n` and use it. If a new number is specified, it will replace the old one as the value of `\%n`.

Now we know what the user wants to dial. In case it was a name instead of a number, we can look it up in our dialing directory. The LOOKUP macro acts as the dialing directory. You give it a name to look up; if it finds the name, it replaces it with the associated number:

```
def lookup -
  if eq \%1 compuserve def \%1 5551423,-
  if eq \%1 telenet    def \%1 5551234,-
  if eq \%1 tymnet     def \%1 5554321
```

(These are not real phone numbers!) As you can see, the lookup macro is just a series of IF EQUAL statements. The DIAL macro uses it like this:

```
lookup \%n
```

This assigns the value of `\%n`, say, `compuserve`, to the first positional macro parameter, `\%1`. If LOOKUP finds this word, it will redefine `\%1` to be a phone number, in this case 5551423. If it doesn't find the word, it will not change the value of `\%1`. This way, you can still give phone numbers in your DIAL commands.

Now let's put all these pieces together into our new DIAL command:

```
def fatal echo \%1\13, stop

def lookup -
  if eq \%1 compuserve def \%1 5551423,-
  if eq \%1 telenet    def \%1 5551234,-
  if eq \%1 tymnet     def \%1 5554321

def dial if = argc 2 assign \%n \%1,-
  if < argc 2 if not def \%n fatal {Dial what?},-
  if > argc 2 fatal {No spaces please.},-
  lookup \%n,-
  take hayes.scr
```

Notice, once again, how commas are used instead of carriage returns to separate Kermit commands within a macro definition, and how the definition can be continued onto multiple lines by using hyphens. And note how the last line of each macro definition does not have a comma or hyphen.

Before you can use these macros, Kermit must execute the DEFINE commands for them. Put the definitions in a file, and then TAKE that file. For example, if the file was called `DIAL.SCR`:

```
MS-Kermit>take dial.scr
```

or put the definitions in your `MSKERMIT.INI` file. To construct your own dialing directory, replace the IF EQUAL lines with similar lines that contain the names and phone numbers of the computers or services that you want to call.

Automated File Transfer

If you've come this far, you're probably asking, "Why not go one step further and automate my entire session with the remote computer?" Good idea. Let's do it.

You've got all the tools you need, but here's a safety tip before we proceed. To have a fully automated session with a remote host or service, you must get Kermit to send it your access codes. But are you going to put your secret password in your script file?

Let's hope not! Anyone who has access to the messy pile of diskettes on your desk could find your password and use it to delete all of your files on the host computer, run up a big bill on your credit card, or worse! So let's design our procedure so that your password does not need to be stored on a disk.

We can put the procedure together from the tools we've created already. But we begin with something new: We prompt the user for the login access codes, so that these codes can be kept in Kermit's memory temporarily rather than on the disk permanently. The command is:

ASK *variable-name prompt*

Prints the prompt. Whatever the user types in response is assigned to the variable, for example:

```
MS-Kermit>ask \%p Password:
Password: secret
MS-Kermit>
```

If you use ASKQ instead of ASK, the user's keystrokes will not echo, a useful safety feature when typing passwords.

We begin by collecting the information we need from the user, in prompting style:

```
set input timeout proceed
echo \13
echo Welcome to the Kermit automat.\13
echo Do as I say and we'll get along fine.\13

:number
ask \%n What number should I dial?\32
if not def \%n goto number
:userid
ask \%u What is your user ID?\32
if not def \%u goto userid
:password
askq \%p What is your password?\32
```

```
if not def \%p goto password
echo
echo \13OK - here we go...\13\10
```

Try these yourself. Notice how the user is required to enter each item. If the user just presses the Enter key, the program issues a prompt for the same item again. Once the phone number, username, and password are collected, the script can dial the phone number:

```
take dial.scr                    ; First define the DIAL macro
dial \%n                         ; Now dial the number
```

Now let's adapt the VAX/VMS login from page 146 to use the username and password variables we have just collected:

```
output \13                       ; Send a carriage return
input 10 Username:               ; Look for Username prompt
if fail errstp {No Username prompt from host}
output \%u\13                    ; Send the username
input 10 Password:               ; Get the password prompt
if fail errstp {No Password prompt from host}
output \%p\13                    ; Send the password
define \%p                       ; Erase the password from memory
input 30 $\32                    ; Get the VMS prompt "$ "
if fail errstp {Login failed}
```

If the script program gets this far, it has logged in to the VAX computer. Now you can have it transfer some files back and forth. Let's assume that your PC is in a branch office of your company, that every night you must send a log (`TODAY.LOG`) of the day's activities to corporate headquarters, and that you also have to fetch a list of new orders to be filled (`ORDERS.LST`). The easiest way to do this is to use the Kermit server on the VAX:

```
output kermit\13                 ; Start Kermit on host
input 10 C-Kermit>               ; Wait for its prompt
if fail errstp {Can't run Kermit on host}
output server\13                 ; Put it in server mode
input 10 reconnect.              ; Get end of server's greeting
if fail errstp {Can't enter server mode on host}
send today.log                   ; Send today's log
if fail errstp {Send TODAY.LOG failed}
get orders.lst                   ; Get new orders
if fail errstp {Get ORDERS.LST failed}
logout                           ; Done, send server away
if exist yesterda.log del yesterda.log
run copy today.log yesterda.log
echo \13All's well that ends well.\13
```

If these script fragments are collected into a file called, say, NIGHTLY.SCR, the simple command TAKE NIGHTLY from the MS-Kermit> prompt will set the entire chain of events into motion. This shows how scripts can be more than a simple convenience: They allow you to automate procedures for unskilled people. A copy of NIGHTLY.SCR could be installed on the PC in each of your branch offices, and every evening the last person to leave could start Kermit and type take nightly.scr at the MS-Kermit> prompt.

These employees don't have to remember numerous commands or learn what they really don't care to know. All they have to do is start their PC, run Kermit, and type one magic command. If that was too hard, you could further automate the process for them by writing a DOS batch file, reducing the process to one single-character DOS command, so that they wouldn't have to know a thing about Kermit:

```
kermit take nightly.scr
```

Call this file X.BAT, and people only have to type X to run the whole procedure. You are not only a hero but probably much wealthier. Remember your friends (and the one who taught you this trick).

Wait, here's one more! Wouldn't it be better if all this uploading took place late at night, when the phone rates were lowest and the corporate mainframe the least busy? Simple: In NIGHTLY.SCR, just before the DIAL command, include a line like:

```
pause 23:59:59
```

This will make Kermit wait until midnight and then dial. No humans need to be present. See, saved you another few bucks!

The Transaction Log

Unattended operations work best when they leave a record of what they have done and what they failed to do. Kermit's method for doing this is called the transaction log. The command that activates transaction logging is (you guessed it):

LOG TRANSACTION *filename*

If you give this command, Kermit will create a file called TRANSACT.LOG that contains a record of all its operations. You can inspect this file in the morning to see whether all went as planned. There is even a command that lets you add your own entries to the transaction log:

WRITE TRANSACTION *item text*

where the *item* is the current TIME, DATE, or PATH, or any TEXT you choose. Any of these items may be followed by additional text. The WRITE command does not add carriage

returns or linefeeds to the log; you have to supply them yourself (\13 is carriage return, \10 is linefeed). This allows you to construct formatted lines containing any combination of things:

```
write transaction date
write trans time Nightly run from
write trans path
write tr text at
write tr time \13\10
```

This will produce a single line like:

```
02-08-1990 Nightly run from C:\OOFA at 23:59:59
```

The WRITE command is also available for the screen (WRITE SCREEN), the session log (WRITE SESSION), and the packet log (WRITE PACKET), for example:

```
write screen text It's\32
write screen time - Do you know where your children are?
```

The \32 is a space.

Let's wrap up our discussion of macros, command files, and scripts by adding transaction logging, plus a little extra bulletproofing, to NIGHTLY.SCR. Here is the complete script program:

```
log trans nightly.log           ; Open the transaction log
if exist today.log goto ok      ; Make sure TODAY.LOG exists
write trans date : TODAY.LOG not found.\13\10
echo TODAY.LOG not found.
stop                            ; Bad news...

:OK                             ; TODAY.LOG exists
take dial.scr                   ; Get DIAL command definition
set input timeout proceed       ; Use IF SUCC/FAIL
echo \13                        ; Begin dialog with user
echo Welcome to the Kermit automat.\13
echo Do as I say and we'll get along fine.\13

:NUMBER                         ; Prompt for phone number
echo
ask \%n What number should I dial?\32
if not def \%n goto number

:USERID                         ; Ask for VAX user ID
ask \%u What is your user ID?\32
if not def \%u goto userid

:PASSWORD                       ; Ask Quietly for Password
askq \%p What is your password?\32
if not def \%p goto password

echo echo\10\13OK - Sleeping till midnight...\13\10
pause 23:59:59                  ; Wait until midnight
```

```
dial \%n                          ; Dial the phone number
def errstp2 echo Error: \%1, hangup,-
 write trans time : \%1, close trans, stop
write trans time Nightly Procedure Begins...\13\10

output \13                        ; Send carriage return to the VAX
input 10 Username:                ; Look for Username prompt
if fail errstp2 {No Username prompt from host}
output \%u\13                     ; Send the username
input 10 Password:                ; Get password prompt
if fail errstp2 {No Password prompt from host}
output \%p\13                     ; Send the password
define \%p                        ; Erase password from memory
input 30 $\32                     ; Get VMS prompt "$ "
if fail errstp2 {Login failed!}

output run kermit\13              ; Start Kermit on host
input 10 C-Kermit>                ; Wait for its prompt
if fail errstp2 {Can't run Kermit on host}
output server\13                  ; Put it in server mode
input 10 reconnect.               ; Get end of greeting
if fail errstp2 {Can't enter server mode on host}
set display quiet                 ; No file transfer screen
send today.log                    ; Send today's log
if fail errstp2 {Send TODAY.LOG failed}
get orders.lst                    ; Get new orders
if fail errstp2 {Get ORDERS.LST failed}
logout                            ; Done, send server away

if exist yesterda.log del yesterda.log
run copy today.log yesterda.log
close trans                       ; Close transaction log
echo \13All's well that ends well.\13
```

There is practically no limit to what you can do with MS-DOS Kermit's command files, macros, and scripts. The techniques illustrated in this chapter can be used to construct procedures to carry on virtually any kind of interaction with a modem or a remote host computer or dialup service. The complexity and length of a command file are limited only by your imagination and the size of your disk. Macros, however, have some limitations that you should keep in mind:

- The maximum length for a macro definition is 255 characters.
- The total memory available for macro names is about 1000 characters, so if you need to define many macros, keep their names short.
- Memory for macro definitions is allocated dynamically from DOS, so space is restricted to the amount of free memory that is not taken up by the Kermit program itself and any other programs that you have loaded. In general, it's wise to abbreviate as much as possible in macro definitions.

Chapter 15

Use of MS-DOS Kermit by People with Disabilities

Unlike most other communication software packages for the PC, MS-DOS Kermit includes features that allow it to be used by people who are blind, deaf, or physically impaired.

Features for People with Visual Impairments

Partially sighted people might be able to read the PC's large-print 40-column display better than its normal 80-column display. For large print, use the DOS command:

```
C>mode 40
```

before starting the MS-DOS Kermit program. Kermit will sense the current screen width and behave appropriately. At the MS-DOS Kermit prompt level, help messages that are longer than 40 characters will wrap to the next line. During terminal emulation, you should inform your host computer that your screen width is 40 rather than 80 so that screen-oriented applications can adjust to your screen size. Two Kermit commands are also useful in this connection:

SET TERMINAL WRAP ON
This ensures that lines of text that are longer than the current screen width are not lost.

SET TERMINAL CURSOR BLOCK
This makes a very big cursor that is much easier to see than Kermit's normal underline cursor.

People who cannot see at all are able to use PCs by installing speech synthesis or Braille devices, which speak or form the words that appear on the screen. For these devices to be useful, text must be displayed on the screen in a linear fashion. Speech synthesis and Braille printing do not work well with applications that use pop-up menus, graphics, colors, or other fancy displays, or that update random fields on the screen as Kermit does during VT terminal emulation or in its file transfer display.

For those who are blind, Kermit can be instructed to write all material to the screen in a way that these devices can interpret sensibly. The commands are:

SET TERMINAL NONE
During terminal emulation, Kermit will ignore all screen formatting and cursor positioning commands from the host, and it will display all arriving characters in linear fashion. For best results, you should also tell the host that you are using a hardcopy (printing) terminal so that it will not send unwanted screen formatting commands at all. See Chapter 8 about setting your terminal type on the host.

SET DISPLAY SERIAL
During file transfer, the status of the file transfer will be written in linear fashion instead of as random updates to a form on the screen.

The effects of these commands can be best appreciated by using Kermit without them.

In addition, the software that drives certain speech synthesizers takes over the PC's keyboard BIOS interrupt. Kermit normally uses the same method during terminal emulation, and this can cause conflicts. To resolve the conflict, use the command:

SET KEY OFF
During terminal emulation, instruct Kermit to access the keyboard using DOS rather than BIOS to give BIOS-level keyboard drivers priority in interpreting your keystrokes. If you don't know what this means, but your special devices are not working right during terminal emulation, try giving this command before you give the CONNECT command. SET KEY OFF does not interfere with your key translations, except that certain keys or combinations (`F11`, `F12`, and `Ctrl-P`, for example) will no longer be recognized by Kermit.

Features for Deaf People

MS-DOS Kermit's "visual bell" option is for those who can't hear the terminal's normal bell. The audible beeps that Kermit normally emits during terminal emulation are converted into screen flashes if you issue the command SET TERMINAL BELL VISIBLE. From then on, if the host sends beeps to get your attention, you will be able to see them.

Features for the Physically Impaired

People who do not have full use of their hands may not be able to type rapidly, and they may find it difficult to press more than one key at a time, which is necessary for entering Shift-, Ctrl-, and Alt-key combinations.

Kermit's SET KEY command lets you assign any character or group of characters to any single key that generates a Kermit scan code. That is, if SHOW KEY prints a scan code when you press the key, you can define that key to transmit anything you like. Frequently used words or key combinations can be assigned to single keys. For example, to assign Ctrl-C to the F1 key, type:

```
MS-Kermit>set key \315 \3
```

You can also assign Kermit verbs to single keys:

```
MS-Kermit>set key \324 \Kexit
```

This assigns the escape back from connect mode function, which is normally *Alt-X* or *Ctrl-]C*, to the single key F10.

These key assignments are effective during terminal emulation but not at the MS-DOS Kermit command line or during file transfer. When typing Kermit commands, or DOS commands in general, mistakes can be corrected by using the Backspace key. During file transfer, the interruption characters *Ctrl-X*, *Ctrl-Z*, *Ctrl-C*, and *Ctrl-E* may be entered by pressing the corresponding single keys X, Z, C, or E (uppercase or lowercase). You may also use IBM's console driver, ANSI.SYS, to reprogram your keys so that difficult key combinations are assigned to single keys that can be used at the MS-Kermit> or DOS prompt.

For general PC use, there are alternate keyboards, switches, foot treadles, joysticks, and special keyboard driver software, to allow data to be input to a computer by people who don't have full use of their hands. Even very simple mechanical keylocks, which hold down the Ctrl, Alt, or Shift keys, can be a big help.

If you are using a special keyboard driver, you can ensure that Kermit does not interfere with it by giving the Kermit command SET KEY OFF.

Chapter 16

MS-DOS Kermit Command Summary

This chapter summarizes MS-DOS Kermit 3.0 commands and features, including those that were not discussed in the text. MS-DOS Kermit can be run interactively, from a batch file, as an "external" DOS command, or with redirected or piped standard input and output. Upon initial startup, the program executes any commands found in the file `MSKERMIT.INI` in the current disk and directory or DOS PATH, or the file specified by `-f` *filename* on the Kermit command line.

Interactive Operation

To run MS-DOS interactively, you invoke the program from DOS command level by typing its name, usually `kermit`. When you see the command's prompt, `MS-Kermit>`, you may type Kermit commands repeatedly until you are ready to exit the program. The command EXIT or QUIT returns you to DOS.

While typing commands, use the Backspace key to erase the character most recently typed, *Ctrl-W* to delete the most recent word, or *Ctrl-U* to delete the entire command, and start the command by pressing the Enter key (carriage return) or *Ctrl-L*. A question mark typed at any point in a command (except in a filename or character string) will give you a brief hint about what's expected at that point. Pressing the Esc key will complete the current word, if possible, and position the cursor at the next command field. If completion is not possible, Kermit will beep. You can cancel any command during its execution by typing *Ctrl-C*.

Piped Operation

MS-DOS Kermit's command-level input and output may be redirected or piped using the normal DOS mechanisms:

```
C>kermit < commands.tak
C>kermit > commands.log
C>kermit < commands.tak > commands.log
C>Kermit < commands.tak | sort > commands.srt
```

Terminal emulation is not affected by redirection or piping.

Command Line Invocation

MS-DOS Kermit may be invoked with command line arguments from DOS command level, for example:

```
C>kermit set port 1, set speed 9600, connect
```

Several commands may be given on the command line, separated by commas. Help and completion are not available when Kermit commands are given on the command line. MS-DOS Kermit will exit back to DOS after completing the specified commands unless you include the STAY command on the command line:

```
C>kermit connect, stay
```

If you want Kermit to execute a different initialization file upon startup, you can create a new file with a unique name and specify its name on the command line:

```
C>kermit -f a:\test\tuesday.ini
```

If you want to run Kermit *au naturel* with all its "factory settings," you can specify the null (empty) initialization file:

```
C>kermit -f nul
```

If `-f` is the only command line option, STAY is implied, and the `MS-Kermit>` prompt will appear.

Remote Operation

MS-DOS Kermit can be put into server mode and accessed from other computers. Furthermore, the MS-DOS CTTY command allows an MS-DOS system to be used from a terminal connected to its communication port, for example CTTY COM1. You can then use DOS and Kermit from a terminal or computer connected to the PC's COM1 device. When MS-DOS Kermit is being run in this manner, you must SET REMOTE ON.

Table 16-1 MS-DOS Kermit Return Codes

ERRORLEVEL	*Kermit Session Status*
0	Entirely successful operation
1	A SEND command completed unsuccessfully
2	A RECEIVE or GET command completed unsuccessfully
4	A REMOTE command completed unsuccessfully
3, 5, 6, 7	Combinations (addition) of the above conditions

Running MS-DOS Kermit in Windowing Environments

You can run MS-DOS Kermit under OS/2, but only in DOS compatibility mode. Separate Kermit programs are available for native OS/2 operation.

You can also run MS-DOS Kermit in DOS windowing environments such as Microsoft Windows and IBM TopView. You must assign Kermit to a window in order for it to run. To make best use of MS-DOS Kermit under Microsoft Windows, copy the file KERMIT.PIF from the original Kermit diskette to the directory on your hard disk where MS-Windows keeps its other Program Information Files (PIFs). If Windows can find Kermit's PIF file, Kermit will work well with mice, accept cut-and-paste material, and shrink to an icon (the word KER in a small box). Use the MS-Windows PIFEDIT program if you want to make changes to the PIF file.

Batch Operation

Like other MS-DOS programs, MS-DOS Kermit may be run from within batch programs by including Kermit command line arguments. If you invoke it without command line arguments, it will run interactively, reading commands from the keyboard and not the batch file. When Kermit exits, batch processing will continue to the end of the batch file. An ERRORLEVEL number is returned by Kermit to assist batch file controls, as shown in Table 16-1.

Here is a sample DOS batch program, SEND.BAT, that runs Kermit to send a file and then reports success or failure to the user:

```
@echo off
if exist %1 goto send
echo File %1 not found
goto end
:send
```

```
echo Sending %1...
kermit -f nul, set display quiet, send %1
if errorlevel 1 goto bad
echo File %1 sent successfully.
goto end
:bad
echo File %1 could not be sent.
:end
```

This batch program can be used to send any file by including the filename on the DOS command line:

```
C>send oofa.txt
```

See the batch section of your DOS manual for details about DOS batch programming.

DOS Environment Variables for Kermit

In your AUTOEXEC.BAT file, you may include a line like:

```
SET KERMIT=command;command;...
```

The commands are special Kermit startup configuration parameters. These include:

INPUT-BUFFER *length*
The number of characters for INPUT and REINPUT commands to retain. The number can be as high as 65535 if your PC has enough memory.

ROLLBACK *length*
The number of screens to retain in the rollback buffer; 10 is the default, and 0 eliminates rollback. The number can be as high as 130 if your PC has enough memory.

Here is a sample AUTOEXEC.BAT entry:

```
set kermit=rollback 20;input-buffer 256
```

File Specifications

In all commands, file specifications may include fully qualified DOS paths, including device specifications. Allowable wildcard characters are * (match from here to end of field) and ? (match a single character), as in DOS. The only difference is that # must be used instead of ? to match the first character of a filename, in order to allow ? to give help in that position. Here are some examples:

```
OOFA.TXT          (File in current disk and directory)
\TOM\OOFA.TXT     (File in directory \TOM\ on current disk)
A:OOFA.TXT        (File in A disk's current directory)
A:\TOM\OOFA.TXT   (File in directory \TOM\ on A disk)
```

```
*.TXT           (All files of type .TXT)
OOFA.*          (All files with name OOFA)
#.*             (All files with a one-character name)
*.?             (All files with a one-character type)
```

Interrupting a File Transfer in Progress

The following keys may be used during a file transfer or execution of a REMOTE command:

X *or* **Ctrl-X**
Stops transferring the current file and goes on to the next one, if any.

Z *or* **Ctrl-Z**
Stops transferring the current file and doesn't send any further files.

E *or* **Ctrl-E**
Sends an error packet to the remote Kermit program to make it quit the current protocol operation.

C *or* **Ctrl-C**
Returns to MS-DOS command level immediately.

Enter
Tries to break a protocol deadlock by retransmitting the most recent packet. Press the Enter key when nothing seems to be happening.

Notation in Command Descriptions

The following notation is used in this command summary:

parameter
Replace this with an actual number, filename, or whatever type of item is called for.

number
Replace with an actual decimal number. In many cases, octal or hexadecimal numbers may also be specified, prefixed by `\o` or `\x`.

filespec
An MS-DOS file specification, which may include disk and directory.

hh:mm:ss
Time of day in 24-hour notation, less than 12 hours from current time.

[*parameter*]
An optional parameter, which may be omitted. (You don't type the brackets.)

{A,B,C}
Choose one of the items listed. Type only one of the listed items. (You don't type the braces or commas.)

[{A,B,C}]
Optionally choose one of the items listed.

text
In examples, text that you type is underlined. When you finish typing the text, press the Enter key.

Ctrl-X
In examples, this means hold down the Ctrl key and press the X key (X can also be any other key). Do not push Enter afterward.

Alt-X
In examples, this means hold down the Alt key and press the X key (X can also be any other key). Do not push Enter afterward.

Ctrl-]X
In examples, this means hold down Ctrl and press the right bracket (]) key, then let go of both these keys and press the X key (X can also be any other key).

Comments and Continuation

In TAKE command files only, commands may have trailing comments preceded by a semicolon, for example:

```
set cursor block  ; I like a big cursor
set term bell vis ; and a visible bell
```

All characters starting with the first semicolon in each line of a TAKE file are ignored by Kermit. To include an actual semicolon in a command within a TAKE file, precede it with a backslash, for example:

```
get oofa.exe\;2
```

Kermit commands can be continued onto the next line by ending the continued line with a hyphen, for example:

```
echo this is not really -
a very long line.
```

Continuation is allowed in command files and macro definitions. If you actually need to end a line with a hyphen, use backslash notation (`\45`).

Table 16-2 MS-DOS Kermit Backslash Notation

Code	*Meaning*
\123	(up to 3 decimal digits)—A decimal number
\d123	(up to 3 decimal digits)—A decimal number
\o123	(up to 3 octal digits)—An octal (base 8) number
\x123	(up to 3 hexadecimal digits)—A hexadecimal (base 16) number
\{}	For grouping, e.g., \{27}3 = ESC 3
\;	Include a semicolon in a TAKE-file command or macro definition
\%	Introduce a Kermit variable, \%1, \%2, ..., \%a, \%b, ... \%z
\K	A Kermit keyboard verb, like \Kexit or \Kbreak
\b	Send a BREAK (OUTPUT command only)
\255	Shorthand for CRLF or LFCR (INPUT command only)
\CD	Carrier Detect RS-232 signal (WAIT command only)
\DSR	Data Set Ready RS-232 signal (WAIT command only)
\CTS	Clear To Send RS-232 signal (WAIT command only)

Backslash Notation

Backslash notation can be used to specify a number or a character in any command where a single character must be specified, such as SET ESCAPE \29. Backslash codes may also be embedded within character strings in the ECHO, INPUT, OUTPUT, ASSIGN, ASK, ASKQ, DEFINE, REINPUT, WRITE, SET KEY, and SET PROMPT commands, for example ECHO \13\10Hello!\7. Any letters following the backslash may be either uppercase or lowercase. Table 16-2 lists MS-DOS Kermit's backslash codes. Whenever there is ambiguity, you can resolve it by using curly braces. The left brace goes immediately after the backslash, and the right brace terminates the backslash code:

```
\{123}                    ASCII character 123 (left brace itself)
\{12}3                    ASCII formfeed (12) followed by 3
\{Kexit}                  Kermit keyboard \Kexit
```

See Table II-3 for a list of Kermit verbs.

MS-DOS Kermit Commands

This section lists all the MS-DOS Kermit commands alphabetically. Subsequent sections give details about the IF, REMOTE, SET, and SHOW commands.

-F *filespec* (from DOS command line only)
Uses *filespec* instead of `MSKERMIT.INI` as the initialization file. `kermit -f nul` runs Kermit without any initialization file at all. The `-f` command can be issued only on the Kermit command line. Example:

```
C>kermit -f midnight.ini
```

ASK *variable-name prompt string*
Prints the prompt on the screen and stores the user's response in the variable. Example:

```
MS-Kermit>ask \%p Password:
Password: secret
MS-Kermit>
```

The value of `\%p` is now "secret" as defined by the user.

ASKQ *variable-name prompt string*
"Ask Quietly"—like ASK, but what user types does not echo. Example:

```
MS-Kermit>ask \%p Password:
Password: _______
MS-Kermit>
```

ASSIGN *variable-name value*
Copies the *value* into the variable. If *value* is a variable itself, its definition is copied rather than its name. Examples:

```
MS-Kermit>assign \%a Hello There!
MS-Kermit>assign \%m \%1
```

BYE
Shuts down a remote Kermit server, logs out its job, and exits from MS-DOS Kermit:

```
MS-Kermit>bye
C>
```

C
Special abbreviation for CONNECT:

```
MS-Kermit>c
```

CD [*path*] (or **CWD** [*path*])
Changes local working directory. If *path* is omitted, this command tells you your current working directory. Unlike the DOS CD command, the Kermit CD command lets you include a disk drive letter and/or a directory name. Examples:

```
MS-Kermit>cd mupeen       (Relative directory)
MS-Kermit>cd \mupeen      (Absolute directory)
MS-Kermit>cd b:           (Disk)
MS-Kermit>cd b:\mupeen    (Disk and directory)
MS-Kermit>cd              (Show current)
```

CLEAR

Clears the communications input buffer. Example:

```
MS-Kermit>cle
```

CLOSE {ALL, PACKETS, SESSION, TRANSACTION}

Closes the specified log file and stops logging (see LOG). Examples:

```
MS-Kermit>close p
MS-Kermit>clo s
MS-Kermit>clo all
```

Logs are closed automatically when you type `exit` at the `MS-Kermit>` prompt.

COMMENT *text*

Does nothing. For adding comments to a TAKE command file. Example:

```
COMMENT - And now set some parameters...
```

CONNECT

Makes a terminal connection to another computer. Type the escape character followed by C to return to the `MS-Kermit>` prompt. The escape character is normally *Ctrl-]* (see Table II-2). Use SET ESCAPE to change the escape character, or use SET KEY to assign `\Kexit` or other Kermit verbs to the keys of your choice. `\Kexit` is also assigned to *Alt-X* on IBM keyboards. (Also see SET TERMINAL.)

DEFINE *name* [*definition*]

Groups MS-DOS Kermit commands together into a macro or defines a variable. If no definition is given, DEFINE undefines the named macro or variable. Commands within the definition are separated by commas, and the commas are translated to carriage returns. Examples:

```
MS-Kermit>define unix set spe 9600, set par none, c
MS-Kermit>define \%n 1-800-555-1234
MS-Kermit>define ibm
```

If the definition includes a variable name, then the variable name is included literally:

```
MS-Kermit>define rename run ren \%1 \%2
```

(compare with ASSIGN).

DELETE *filespec*

Deletes local PC files. Example:

```
MS-Kermit>delete junk.*
```

DIRECTORY [*filespec*]
Lists names, sizes, and dates of local PC files. Runs DOS's DIR command. Examples:

```
MS-Kermit>directory
MS-Kermit>directory oofa.*
MS-Kermit>dir a:
MS-Kermit>dir \chris\
MS-Kermit>dir a:\chris
```

DISABLE *function*
Restricts the specified function as indicated when in server mode: CD (disallow), DEL (current directory only), DIR (current directory), FIN (disallow FINISH, BYE, and LOGOUT), GET (current directory), HOST (disallow), KERMIT (REMOTE KERMIT commands not allowed), LOGIN (not required), SEND (current directory), SPACE (disallowed), TYPE (current directory). ALL disables all these at once. Example:

```
MS-Kermit>disable all
```

[DO] *macro-name parameters*
Performs the commands in a macro, with parameters for substitution. The word DO can be omitted; you can just type the name of the macro. Examples:

```
MS-Kermit>do ibm
MS-Kermit>do dial 7654321
MS-Kermit>ibm
MS-Kermit>dial 7654321
```

Parameters (the words after the macro name) are automatically assigned to variables `\%1`, `\%2`, and so on up to `\%9`.

ECHO *text*
Displays the *text* on the screen and interprets any backslash-escapes within the text. Example:

```
MS-Kermit>echo \13\10\10\7Wake up, Bill!!\7\10\10\13
```

ENABLE *function*
Allows the specified function as indicated when in server mode: CD (to any directory), DEL (any file anywhere), DIR (of any directory), FIN (FINISH, BYE, and LOGOUT), GET (file from any directory), HOST (commands allowed), LOGIN (required), SEND (to any directory), SPACE (allowed), TYPE (files in any directory). ALL enables all these at once. Example:

```
MS-Kermit>ena login
```

EXIT
Exits from MS-DOS Kermit. Does not close the connection by hanging up. Example:

```
MS-Kermit>ex
C>
```

FINISH
Shuts down a remote Kermit server but does not exit from MS-DOS Kermit and does not terminate the remote host session (compare with BYE). Example:

```
MS-Kermit>fin
MS-Kermit>
```

GET *remote-filespec*
Asks a Kermit server to send the specified file(s). Use this command instead of RECEIVE when communicating with a Kermit server. To specify an alternate name to store the file under when it arrives, type get followed immediately by the Enter key to be prompted for remote and local filespecs separately. Examples:

```
MS-Kermit>get oofa.txt
MS-Kermit>get *.c
MS-Kermit>get
 Remote Source File: oofa.txt
 Local Destination File: oofa.new
```

GOTO *label*
In script execution, transfers control to the specified label. The label must appear in the left margin, beginning with a colon (:), and it must be within the same macro definition or, if the GOTO is not in a macro definition, within the same TAKE file. Example:

```
goto xxx
echo You won't see this!
:xxx
echo You will see this.
```

HANGUP
Instructs the modem to hang up the phone by briefly turning off the DTR modem signal (see Glossary) on the current port or to close the network connection. Example:

```
MS-Kermit>hang
```

HELP
Displays a brief help message about MS-DOS Kermit:

```
MS-Kermit>help
```

I
Special abbreviation for INPUT. Example:

```
MS-Kermit>i 10 login:
```

IF [NOT] *condition command*
In a script, executes the command if the condition is true or, if NOT is included, executes the command if the condition is not true (listed below).

INPUT [*timeout*] *text*
Tries to receive the specified text from the remote system within *timeout* seconds. Sets SUCCESS or FAILURE for subsequent IF commands. *text* may contain backslash codes. Examples:

```
input 5 Password:\32
in Password:\32
```

Also see SET INPUT.

LOG PACKETS [*filespec*]
Records file transfer packets in the specified file or device. Default file is PACKET.LOG. Example:

```
MS-Kermit>log packets julie.log
```

LOG SESSION *filespec*
Records your terminal session in the specified file or device. Default file is SESSION.LOG. Example:

```
MS-Kermit>log sess monday.log
```

LOG TRANSACTION *filespec*
Reports the progress of file transfers in the specified file or device. Default file is TRANSACT.LOG. Example:

```
MS-Kermit>log t overnite.log
```

LOGOUT
Shuts down the remote server, logs out the remote host session, but doesn't exit from MS-DOS Kermit (compare with BYE, FINISH). Example:

```
MS-Kermit>logo
```

MAIL *filespec address*
Sends the file as mail to the specified address. The remote Kermit must be in RECEIVE or SERVER mode, and must support the MAIL command. The fields of this command can be typed on separate lines. Examples:

```
MS-Kermit>mail message.txt michael@cuvma.bitnet
MS-Kermit>mail
 Local Source File: message.txt
 To: michael@cuvma.bitnet
```

OUTPUT *text*
Sends the specified text to the remote host as if you had typed it; *text* may contain backslash codes. Example:

```
output run kermit\13
```

PAUSE [{*seconds, hh:mm:ss*}]
Sleeps for the specified number of seconds or until the specified time of day, which cannot be more than 12 hours from the current time. Default interval is one second. Examples:

```
MS-Kermit>pause
MS-Kermit>pause 3
MS-Kermit>pause 23:59:59
```

Can be interrupted by typing any key. Turns on the DTR signal (see Glossary).

POP
When executing a command file or macro, returns immediately to the invoking level (macro, TAKE file, or `MS-Kermit>` prompt). POP does nothing at command level. Example:

```
if failure pop
```

PUSH
Invokes an MS-DOS command processor "underneath" Kermit. Type `EXIT` at the DOS prompt to return to Kermit. Example:

```
MS-Kermit>push
C>exit
MS-Kermit>
```

QUIT
Synonym for EXIT (see EXIT). Example:

```
MS-Kermit>q
```

R
Special abbreviation for RECEIVE:

```
MS-Kermit>r
```

RECEIVE [*alternate-name*]
Waits for files from the other Kermit, which must be given a SEND command. If *alternate-name* is given, renames the first arriving file to this name. *alternate-name* may be any combination of disk, directory, and filename. If *alternate-name* is only a disk and/or directory, then all arriving files will be placed there under their own names. Examples:

```
MS-Kermit>receive
MS-Kermit>rec newname.txt
MS-Kermit>rec a:
MS-Kermit>r \chris\
MS-Kermit>r a:\chris\
```

REMOTE *command*
Prefix for commands to remote Kermit server (listed below).

REPLAY *filename*

Replays a session log through the terminal emulator to relive a terminal session. Terminal type must be set the same as it was during logging. Example:

```
MS-Kermit>set term tek
MS-Kermit>replay usa.tek
```

RUN *command* [*arguments*]

Invokes an MS-DOS command or program, with any command-line arguments that may be given. Examples:

```
MS-Kermit>run basic
MS-Kermit>run edlin autoexec.bat
MS-Kermit>run ren oldname.txt newname.txt
```

S

Special abbreviation for SEND:

```
MS-Kermit>s oofa.txt
```

SEND *filename* [*alternate-name*]

Sends files to remote Kermit receiver or server. If *alternate-name* is given, sends the first file using that name. If the SEND command is typed on a line by itself, you will be prompted separately for the two filenames. Examples:

```
MS-Kermit>send oofa.txt
MS-Kermit>sen oofa.txt newname.txt
MS-Kermit>s *.txt
MS-Kermit>send
 Local source file: oofa.txt
 Remote destination file: newname.txt
```

SERVER [{*seconds*, *hh:mm:ss*}]

Puts MS-DOS Kermit into SERVER mode. The MS-DOS Kermit server honors the following requests, within the restrictions established by the ENABLE and DISABLE commands (see DISABLE, ENABLE, and REMOTE):

SEND	REMOTE CD	REMOTE LOGIN
GET	REMOTE DELETE	REMOTE MESSAGE
FINISH	REMOTE DIR	REMOTE SEND
BYE	REMOTE HELP	REMOTE SET
LOGOUT	REMOTE HOST	REMOTE SPACE
	REMOTE KERMIT SET	REMOTE TYPE

The MS-DOS Kermit server can be run indefinitely (until it receives a BYE or FINISH command), or for the specified number of seconds, or until the specified time (unless a BYE or FINISH command comes first and FIN is not DISABLEd). Examples:

```
MS-Kermit>server                (Serve forever)
MS-Kermit>server 3600           (For one hour)
MS-Kermit>server 20:00:00       (Till 8:00 P.M.)
```

After giving the SERVER command, you can get back to the MS-Kermit> prompt by typing Ctrl-C or just the letter C (also see SET SERVER, ENABLE, and DISABLE).

SET *parameter value*
Sets various parameters (listed below).

SHOW *topic*
Displays settings or other information related to the given topic (see below).

SPACE [*disk*]
Reports the available disk space on the current or specified disk. Examples:

```
MS-Kermit>space
MS-Kermit>space a:
```

STATUS
Shows values of all SET parameters. Example:

```
MS-Kermit>stat
```

STAY
(from DOS command line only) Stays within Kermit after the other commands have executed. Example:

```
C>kermit connect, stay
```

STOP
In a command file or macro, returns directly to MS-Kermit> prompt level (or exit to DOS if invoked with command line arguments). Example:

```
if alarm stop
```

TAKE *filespec*
Executes MS-DOS Kermit commands from the specified file. A TAKE file may contain any MS-DOS Kermit commands, including other TAKE commands. Example:

```
MS-Kermit>take wp.ini
```

TRANSMIT *filespec* [*prompt*]
Sends the characters from the file to the host with no error checking, just as if you were typing them (but faster). The *prompt* is the character to wait for from the host before sending the next line. The default prompt is linefeed (\10) or the current handshake character, if any (SET HANDSHAKE). Use \0 to tell MS-DOS Kermit not to wait for any prompt character at all. (Also see SET TRANSMIT.) Example:

```
MS-Kermit>transmit oofa.txt
MS-Kermit>tr oofa.txt \0
```

TYPE *filespec*
Displays a local file on the screen. Example:

```
MS-Kermit>type \autoexec.bat
```

The Kermit TYPE command just runs the DOS TYPE command, so long files will fly by faster than you can read them unless you type *Ctrl-S* and *Ctrl-Q* to stop and resume the display. To pause automatically after each screenful, use the DOS MORE command. Example:

```
MS-Kermit>run more < \autoexec.bat
```

VERSION

Displays the MS-DOS Kermit program version number and release date. Example:

```
MS-Kermit>v
IBM-PC MS-Kermit: 3.0, 8 Feb 90
```

WAIT {*seconds, hh:mm:ss*} **[\CD] [\DSR] [\CTS]**

Waits the specified number of seconds or until the specified time for the given modem signal(s) to appear. If all the given signals do not appear within the allotted time, set the FAILURE flag. Default time to wait is one second. If no modem signals are specified, WAIT is equivalent to PAUSE. Examples:

```
MS-Kermit>wait
MS-Kermit>wait \cd
MS-Kermit>wait 5 \cd
MS-Kermit>wait 10 \cd \dsr
MS-Kermit>wait 03:21:43 \cts
```

WRITE {PACKET, SESSION, SCREEN, TRANSACTION} *object* [*text*]

Writes the specified object to the specified log (if open) or to the screen. Objects are DATE (current date), INPUT-BUFFER (contents of port input buffer), PATH (current disk and directory), TEXT (the characters that follow the word TEXT), TIME (current time), VERSION (MS-DOS Kermit's program version number), ARGC (value of the ARGC variable), COUNT (value of the COUNT variable), and ERRORLEVEL (value of the ERRORLEVEL variable). Text may follow any of these items. A space is written after each of these objects except TEXT, and no carriage returns or linefeeds are written unless you include them in the text. Example:

```
write session text Begin session log at\32
wri ses date
wri ses time \13\10
```

produces a line like this:

```
Begin session log at 10-22-1989 14:11:00.27
```

You can use WRITE SCREEN in TAKE files and macros to display the date, time, and other objects during execution.

IF Commands

The general form of the IF command is:

IF *condition command*

If the *condition* is true, then the *command* is executed. The word NOT can be placed before any of the IF conditions to reverse the meaning:

IF NOT *condition command*

which means that the *command* will be executed if the *condition* is *not* true. Here are MS-DOS Kermit's IF commands:

IF ALARM *command*
The *command* is executed if the SET ALARM time has passed. Example:

```
set alarm 180  ; A 3-minute egg timer
run cls        ; clear screen
:loop
write screen time \13
pause 1
if not alarm goto loop
echo \7Your egg is ready!\7\13
```

IF COUNT *command*
Subtracts one from COUNT; if COUNT is greater than zero (see SET COUNT), the *command* is executed. Example:

```
set count 5
run cls
:loop
write screen count
pause 1
if count goto loop
echo \13\7Blast off!\7
```

IF DEFINED *name command*
The *command* is executed if the macro or variable with the given *name* is defined (see DEFINE). Example:

```
if defined \%1 assign \%a \%1
```

IF ERRORLEVEL *number command*
The *command* is executed if the DOS ERRORLEVEL matches or exceeds the given *number*. Example:

```
if errorlevel 13 stop
```

IF EQUAL *word1 word2 command*
The *command* is executed if the character string *word1* is the same as *word2*. *word1* and *word2* may be variables. Alphabetic case is treated according to SET INPUT CASE. Example:

```
set input case ignore
ask \%a Should I stop? (yes or no)
if equal \%a yes stop
```

IF = *number1 number2 command*
The *command* is executed if *number1* is equal to *number2*. *number1* or *number2* may be numeric constants or variables with numeric values, including COUNT, ARGC, and ERRORLEVEL. Example:

```
if = \%1 3 echo The value is 3
if = ARGC 2 goto ok
if not = ARGC 2 goto bad
```

Note: ARGC is the number of words in a macro invocation, for example, in

```
dial 7654321
```

ARGC is 2.

IF > *number1 number2 command*
The *command* is executed if *number1* is greater than *number2*. *number1* or *number2* may be numeric constants or variables with numeric values, including COUNT, ARGC, and ERRORLEVEL. Examples:

```
if > ARGC 3 echo Too many arguments!
if not > \%n 4 echo \%n is less than or equal to 4.
```

IF < *number1 number2 command*
The *command* is executed if *number1* is less than *number2*. *number1* or *number2* may be numeric constants or variables with numeric values, including COUNT, ARGC, and ERRORLEVEL. Example:

```
if < COUNT 5 echo Still counting...\13
```

Note: When COUNT is used with `IF =`, `IF <`, or `IF >`, its value is not changed (see IF COUNT).

IF EXIST *filename command*
The *command* is executed if the given file exists. Example:

```
if exist \autoexec.bat run ren \autoexec.bat \a.tmp
```

IF FAILURE *command*
The *command* is executed if the most recent INPUT, WAIT, SEND, RECEIVE, GET, BYE, FINISH, or LOGOUT command failed. For INPUT, the IF FAILURE command is executed only if SET INPUT TIMEOUT PROCEED is in effect. Example:

```
set input timout proceed
input 10 login:
if failure goto bad
echo Got login prompt!\13
goto good
:bad
echo Failed to get login prompt.\13
stop
:good
```

IF SUCCESS *command*
The *command* is executed if the most recent INPUT, WAIT, SEND, RECEIVE, GET, BYE, FINISH, or LOGOUT command succeeded. For INPUT, this command is executed only if SET INPUT TIMEOUT PROCEED is in force. Example:

```
set input timout proceed
input 10 login:
if success goto good
echo Failed to get login prompt.\13
stop
:good
echo Got login prompt!\13
```

REMOTE Commands

The following commands can be used only when communicating with a remote Kermit server. Results, if any, are displayed on the screen. Or if *> filespec* is added to the end of the command, the results are written to the specified file. Any REMOTE command can have its results redirected.

REMOTE CD [*directory* [*password*]]
(also REMOTE CWD) Changes current directory on the remote host. If *directory* not specified, changes to default directory. Examples:

```
MS-Kermit>remote cd /usr/michele
MS-Kermit>remote cd
```

REMOTE DELETE *filespec*
Deletes remote file(s). Example:

```
MS-Kermit>remote delete $disk1:[dave]*.tmp
```

REMOTE DIRECTORY [*filespec*]
Lists remote file(s). Examples:

```
MS-Kermit>remote directory
MS-Kermit>rem dir $disk1:[rose]oofa.*
MS-Kermit>remote directory > remote.dir
```

REMOTE HELP
Asks the server to list the services it provides:

```
MS-Kermit>remo help
```

REMOTE HOST *command*
Sends a command to the remote host in its own command language, passed through the remote Kermit server, which sends the results back. The command must not be an interactive command. Examples:

```
MS-Kermit>rem host cp q names
MS-Kermit>rem host grep -i kermit *
```

REMOTE KERMIT *command*
Sends a command to the remote Kermit server in its own command language. Example:

```
MS-Kermit>rem kermit set file type binary
```

REMOTE LOGIN [*user password*]
Logs in to a remote Kermit server. If *user* and *password* are omitted from the command line, you will be prompted for them. If the password is given on the REMOTE LOGIN command line, it will echo. If you're prompted for it, it will not echo. Examples:

```
MS-Kermit>rem login vincent secret
MS-Kermit>rem login
 Username: vincent
 Password: ______
 Account:
```

In response to the `Account:` prompt, type your account if one is required, or just press the Enter key.

REMOTE MESSAGE *text*
Sends a one-line message to be displayed on the remote Kermit server's screen. Example:

```
MS-Kermit>rem message Hello Henry!
```

REMOTE SET *parameter value*
Tells the remote Kermit server to change one of its settings. Example:

```
MS-Kermit>rem set file type binary
```

REMOTE SPACE [*area*]
Shows available disk space on remote host in the current device or directory or the one you specify. Examples:

```
MS-Kermit>rem space
MS-Kermit>rem space $disk1:[gary]
```

REMOTE TYPE *filespec*
Displays remote file(s) on your PC screen. Example:

```
MS-Kermit>rem type oofa.txt
```

REMOTE WHO [*user*]
Displays users who are logged in to the remote system or information about the specified user. Examples:

```
MS-Kermit>remote who
MS-Kermit>rem who annette
```

SET Commands

SET ALARM {*seconds, hh:mm:ss*}
In scripts, sets an alarm (for use with IF ALARM). Examples:

```
MS-Kermit>set alarm 10
MS-Kermit>set alarm 22:00:00
```

SET ATTRIBUTES {ON, OFF}
Enables or disables the use of file attribute packets. Attribute packets are used by the file sender to tell the receiver the file's size, date, type, and so forth. If Kermit refuses to transfer a file for reasons that you believe are unjustified, you can disable the use of attribute packets like this:

```
MS-Kermit>set attr off
```

SET BAUD *number*
Synonym for SET SPEED. Example:

```
MS-Kermit>set baud 2400
```

SET BELL {ON, OFF}
Whether to beep at the end of a file transfer. Unless told otherwise, Kermit will beep. Example:

```
MS-Kermit>set bell off
```

SET BLOCK-CHECK-TYPE {1, 2, 3}
Level of error checking for file transfer. Type 1 is a 6-bit checksum, type 2 is a 12-bit checksum, type 3 is a 16-bit cyclic redundancy check (CRC). The higher the type, the more effective the error checking. Type 1 is used by default. Example:

```
MS-Kermit>set block 3
```

SET COUNT *number*
In scripts, sets up a loop counter (for use with IF COUNT). Example:

```
set count 3
:loop
echo hello\13
if count goto loop
echo goodbye\13
```

SET DEBUG {ON, OFF, PACKETS, SESSION}

Displays PACKETS during file transfer; displays control and 8-bit characters specially during terminal SESSION. ON means both PACKETS and SESSION. OFF means no debugging. Examples:

```
MS-Kermit>set debug packets
MS-Kermit>set deb ses
MS-Kermit>set deb off
```

SET DEFAULT-DISK *disk-name*

Default disk drive for sending and receiving files. Equivalent to CD *disk-name*. Example:

```
MS-Kermit>set def a:
```

SET DELAY *seconds*

In remote mode, the number of seconds to pause after a SEND command before sending the file. Example:

```
MS-Kermit>set remote on
MS-Kermit>set delay 5
MS-Kermit>send oofa.txt
```

SET DESTINATION {DISK, PRINTER, SCREEN}

Default destination device for incoming files. Examples:

```
MS-Kermit>set destination printer
MS-Kermit>set dest screen
```

SET DISPLAY {QUIET, REGULAR, SERIAL, 7-BIT, 8-BIT}

For selecting the type of file transfer display and terminal screen displays. QUIET means no file transfer display at all; REGULAR means a continuously updated screen form; SERIAL is for use with hardcopy or Braille terminals or speech synthesizers. 7-BIT and 8-BIT control the display of characters on the screen during terminal emulation: 7-BIT means the 8th bit of incoming characters should be stripped before the character is displayed; 8-BIT means all 8 bits are displayed if PARITY is NONE. Examples:

```
MS-Kermit>set display quiet
MS-Kermit>set disp serial
MS-Kermit>set di 8
```

SET DUMP *filespec*

Specifies screen-copy (screen dump) filename. KERMIT.SCN is the default. Example:

```
MS-Kermit>set dump rick.scn
```

SET DUPLEX {FULL, HALF}

FULL means remote echo and Xon/Xoff flow control; HALF means local echo and RTS/CTS flow control. The default is FULL. Example:

```
MS-Kermit>set dup h
```

SET EOF {CTRL-Z, NOCTRL-Z}

Method for determining or marking the end of a PC file during file transfer. NOCTRL-Z (the default) means the end of file is its last character. CTRL-Z means the end of a file is marked by a Ctrl-Z character, even if it is not the last character in the file. Example:

```
MS-Kermit>set eof ctrl-z
```

SET ERRORLEVEL *number*

Status code to be returned by Kermit upon exit, for use by DOS batch. Kermit normally sets its status code automatically according to the success or failure of its SEND, RECEIVE, GET, and REMOTE commands (see Table 16-1). Example:

```
MS-Kermit>set err 3
```

SET ESCAPE *character*

Escape character for CONNECT, normally *Ctrl-]* (\29). The character may be typed literally or entered using backslash notation. Example:

```
MS-Kermit>set esc \28
```

SET FILE {CHARACTER-SET, TYPE, WARNING}

Sets file-related parameters (examine them with SHOW FILE).

SET FILE CHARACTER-SET {CP437, CP850, CP860, CP863, CP865}

Tells MS-DOS Kermit which IBM code page to use when translating a text file during file transfer (see Table II-5); by default it is your current code page. When sending a file, the file character set is translated into the transfer character set (see SET TRANSFER) if the file type is TEXT, and when receiving the transfer character set is translated to the file character set. Example:

```
MS-Kermit>set file char cp860
```

This command has significance only for text files that contain special national characters. ASCII text files are the same in any code page.

SET FILE TYPE {BINARY, TEXT}

If the file type is BINARY, no translations are done during file transfer. If the file type is TEXT, Kermit translates between the current transfer and file character sets. The default file type is TEXT. Use SET FILE TYPE BINARY to transfer .EXE files, application-specific files (databases, spreadsheets, and so on).

```
MS-Kermit>set file type bin
```

SET FILE WARNING {ON, OFF}

Specifies how to handle filename collisions. If ON (the default), an arriving file that has the same name as an existing file will be given a new and unique name. If OFF, arriving files will overwrite existing files of the same name.

```
MS-Kermit>set fil w on
```

SET FLOW-CONTROL {NONE, XON/XOFF}
Selects the full-duplex flow control method. Xon/Xoff is the default. Example:

```
MS-Kermit>set flo none
```

SET HANDSHAKE {XON, BELL, ESC, CR, LF, NONE, CODE *ascii-code*}
Half-duplex line turnaround character. To be used during file transfer and with the TRANSMIT command. Examples:

```
MS-Kermit>set handshake xon
MS-Kermit>set handsh code 25
MS-Kermit>set ha none
```

SET INCOMPLETE {DISCARD, KEEP}
What to do with an incompletely received file. The default is DISCARD. Example:

```
MS-Kermit>set inc keep
```

SET INPUT *parameter value*
Various parameters for the INPUT script command:

SET INPUT CASE {IGNORE, OBSERVE}
Whether to ignore or observe alphabetic case when scanning arriving characters for INPUT text. Case is observed by default. Also applies to IF EQUAL. Example:

```
MS-Kermit>set inp case ign
```

SET INPUT DEFAULT-TIMEOUT *seconds*
How many seconds to wait for the specified input if a timeout interval is not specified. The default interval is one second. Example:

```
MS-Kermit>set inp def 5
MS-Kermit>input login:
```

SET INPUT ECHO {ON, OFF}
Whether the INPUT command should display characters on the screen as it reads them. Normally, the characters are displayed. Example:

```
MS-Kermit>set inp e off
```

SET INPUT TIMEOUT-ACTION {PROCEED, QUIT}
Whether MS-DOS Kermit should proceed to the next statement in a macro or command file if an INPUT command fails to read the specified characters, or else quit from the command file or macro. By default, Kermit proceeds. Example:

```
MS-Kermit>set inp tim quit
```

SET KEY [*keycode definition*]
Specify key redefinitions or keystroke macros (see below).

SET LOCAL-ECHO {ON, OFF}

Specifies whether MS-DOS Kermit should echo characters itself during terminal emulation (ON) or let the remote host echo them (OFF). SET LOCAL-ECHO ON is implied by SET DUPLEX HALF, and SET LOCAL-ECHO OFF is implied by SET DUPLEX FULL. The default is OFF, for full-duplex remote echoing. Example:

```
MS-Kermit>set loc on
```

SET LOG

Synonym for LOG (See LOG). Example:

```
MS-Kermit>set log packets p.log
```

SET MODE-LINE {ON, OFF}

Whether to display a mode line at the bottom of the screen during terminal emulation. Normally, the mode line is displayed. Example:

```
MS-Kermit>set mode off
```

SET PARITY {NONE, EVEN, ODD, MARK, SPACE}

Character parity to use during terminal emulation and file transfer. If NONE, 8-bit data can be transferred; otherwise, only 7-bit characters can be used during terminal emulation, and a special prefixing mechanism is used for 8-bit data. Example:

```
MS-Kermit>set par even
```

SET PORT {*port, network, etc.*}

Selects a port, driver, or network for communication. The default port is COM1.

SET PORT COM1

(or COM2, COM3, or COM4) Selects a regular IBM or IBM-compatible communications port or internal modem.

SET PORT 1

(or 2, 3, or 4) Equivalent to SET PORT COM1, COM2, COM3, or COM4. Example:

```
MS-Kermit>set port 2
```

SET PORT BIOS1

(or BIOS2, BIOS3, or BIOS4) Selects communications port 1, 2, 3, or 4 but goes through the system BIOS. This allows use of Kermit through any kind of communications device that is supported by a BIOS-level COM driver. Examples include nonstandard internal modems and certain local area networks. Example:

```
MS-Kermit>set port bios1
```

SET PORT DECNET

DECnet/DOS users can use this command to communicate with a VAX/VMS or other DECnet host at very high speeds over the PC's network interface. Both CTERM and

LAT[22] protocols are supported. You must specify the node name of the DECnet host you wish to connect to:

```
MS-Kermit>set port dec vaxine
```

If it is a LAT connection that requires a password, you may include that too:

```
MS-Kermit>set port dec cumin secret
```

If you leave out the node name, Kermit will use the one you gave in your most recent SET PORT DECNET command.

SET PORT NETBIOS

Users of networks like IBM Token Ring or PC Network can communicate with each other at high speeds through the Netbios interface. For PC-to-PC connections, only file transfer is supported. Terminal emulation is possible with hosts that support Netbios terminal connections (AT&T Starlan is an example). To set your PC up as a network server, issue the command:

```
MS-Kermit>set port netbios
```

This will automatically assign a network name to your Kermit process, which is the same as your computer's network name, with a `.K` added to the end. In the example the computer's network name is `Lisa`:

```
MS-Kermit>set port netbios
Checking to see if node name is unique...
Node name: Lisa.K
```

If your PC does not have a network name, Kermit will use `mskermit.K` (lowercase mskermit, uppercase K). If other PCs on the network have the same name, Kermit will prompt you to enter a new name. Other network users can use MS-DOS Kermit to transfer files with your Kermit program by using this name:

```
MS-Kermit>set port netbios lisa.k
```

Remember, the PC you are trying to access must already be running Kermit and must have already been given the SET PORT NET command.

SET PORT NOVELL

For accessing Novell asynchronous communication servers. Example:

```
MS-Kermit>set port nov
```

After you CONNECT, you will have a dialog with the Novell server to select the desired service.

---

[22]CTERM is the protocol used with the DECnet SET HOST command, and LAT (Local Area Transport) is the protocol used by DECserver terminal servers.

SET PORT OPENNET [*name*]
For Intel OpenNET. Works like SET PORT NETBIOS. Example:

```
MS-Kermit>set port opennet connie.K
```

SET PORT UB-NET1
For Ungermann-Bass Net/One. Example:

```
MS-Kermit>set port ub
```

SET PROMPT *string*
Changes the `MS-Kermit>` prompt to *string*. Example:

```
MS-Kermit>set prompt Jeannette>
Jeannette>
```

SET RECEIVE *parameter value*
Requests the remote Kermit to use the specified parameters (listed below).

SET REMOTE {ON, OFF}
Use SET REMOTE ON when running MS-DOS Kermit interactively from a terminal or PC connected to the PC's communication port, which is done using the CTTY COM1 command of DOS. SET REMOTE ON turns off Kermit's file transfer display so it won't interfere with the packets that are being sent. Example:

```
MS-Kermit>set rem on
```

SET RETRY *number*
Packet retransmission threshold. Normally, Kermit will try to send a particular packet up to five times before giving up. Use this command to raise or lower this number. Example:

```
MS-Kermit>set ret 20
```

SET SEND *parameter value*
Use the specified parameters during file transfer (listed below).

SET SERVER LOGIN *user password*
Establishes a username and password that must be sent by the REMOTE LOGIN command before the server will respond to any other requests. Example:

```
MS-Kermit>set server login milly xyzzy
```

SET SERVER TIMEOUT *seconds*
How often the MS-DOS Kermit server times out between commands, normally zero, meaning no timeout at all between commands. Example:

```
MS-Kermit>set serv tim 30
```

SET SPEED *number*
Communications port line speed in bits per second. Permissible values include 110, 300, 1200, 2400, 4800, 9600, 19200, 38400, and 57600. Example:

```
MS-Kermit>set sp 9600
```

SET TAKE-ECHO {ON, OFF}
Specifies whether commands from TAKE files and macros are echoed on your screen during execution. Normally OFF. If ON, the commands will appear on your screen. Example:

```
MS-Kermit>set tak on
```

SET TERMINAL *parameter value*
Emulation and parameters (listed below).

SET TIMER {ON, OFF}
Enable or disable timeouts and automatic packet retransmission during file transfer. Normally ON. Example:

```
MS-Kermit>set tim off
```

SET TRANSFER CHARACTER-SET {LATIN1, ASCII, TRANSPARENT}
Specifies the character set to be used for file transfer. LATIN1 means to translate between the current file character set (see SET FILE) and ISO Latin Alphabet 1. ASCII means to use only 7-bit ASCII characters during file transfer. TRANSPARENT means not to translate characters at all. The default is TRANSPARENT. Example:

```
MS-Kermit>set transf char latin1
```

SET TRANSLATION INPUT *port-char screen-char*
Translates the specified arriving port character to the specified screen character. SET TRANSLATION INPUT {ON, OFF} enables or disables translations entered by this method. They are normally disabled. (See Tables II-4 and II-5.) Example:

```
MS-Kermit>set transl inp \91 \132
```

SET TRANSMIT *parameter value*
Controls the behavior of the TRANSMIT command (see TRANSMIT). Normally, TRANSMIT sends a text file a line at a time and strips the linefeed (LF) from the end of the line, sending only the carriage return (CR, same as Enter), just as you would type it, and waiting for the host to echo LF before sending the next line. (The echoed LF is called the *prompt*.)

SET TRANSMIT FILL-EMPTY-LINE {NONE, SPACE, *char***}**
Normally, a blank line is sent as a single CR. Some hosts treat blank lines as end of file. You can have Kermit add a character to each blank line so that such hosts will accept them. Examples:

```
MS-Kermit>set transm empty space
MS-Kermit>set transm em X
```

SET TRANSMIT LINE-FEEDS-SENT {ON, OFF}
Tells Kermit to send both the CR and the LF at the end of each line, rather than just the LF:

```
MS-Kermit>set transm line on
```

SET TRANSMIT PROMPT *char*
Changes the default prompt for TRANSMIT from LF to whatever your host application is using. For example, if you are transmitting into a text editor whose prompt ends with a question mark:

```
MS-Kermit>set transm prompt \63
```

Use SET TRANSMIT PROMPT `\0` to tell Kermit not to wait for any prompt at all.

SET WARNING {ON, OFF}
Synonym for SET FILE WARNING (see SET FILE WARNING). Example:

```
MS-Kermit>set warn on
```

SET WINDOW *number*
Specifies packet window size, 1 to 31, for use only on full-duplex connections. Improves speed of file transfer over long-distance connections. Example:

```
MS-Kermit>set win 6
```

SET SEND and SET RECEIVE Commands

The SET RECEIVE commands tell MS-DOS Kermit to tell the other Kermit what parameters to use during file transfer. The SET SEND commands tell MS-DOS Kermit to use the given parameters when sending packets, even if the other Kermit asks for something else:

SET {SEND, RECEIVE} END-OF-PACKET *char*
Packet terminator to use, normally carriage return (`\13`). Change this only if carriage return does not work. Example:

```
MS-Kermit>set rec end \27
```

SET {SEND, RECEIVE} PACKET-LENGTH *number*
Maximum packet length. SET REC PACK 94 or greater enables long packets; SET SEND PACK *xx* overrides the negotiated length, but only if *xx* is shorter. Longer packets will speed up file transfer if the connection is not noisy. MS-DOS Kermit's maximum length is 2000. Example:

```
MS-Kermit>set rec pack 1000
```

SET {SEND, RECEIVE} PADCHAR *character*
Prepacket padding character to use. Rarely needed. Example:

```
MS-Kermit>set send padc \127
```

SET {SEND, RECEIVE} PADDING *number*
Number of padding characters to send (SET SEND) or to request (SET RECEIVE) per packet, normally zero. Rarely needed. Example:

```
MS-Kermit>set rec padd 3
```

SET SEND PAUSE *number*
Interpacket pause in milliseconds (thousandths of seconds). Only for sending. Example:

```
MS-Kermit>set send pau 100
```

SET {SEND, RECEIVE} QUOTE *character*
Control-character prefix to use when sending packets. Normally `#`. Should never need to be changed.

SET {SEND, RECEIVE} START-OF-PACKET *character*
Control character that marks the beginning of a packet. Normally Ctrl-A (`\1`). Change this if the Ctrl-A character is intercepted by some device (like a modem) between your PC and the other computer. You must change the start-of-packet character in *both* places. In this example, the packets that MS-DOS Kermit sends are changed to begin with a Ctrl-B character. The packets that the other Kermit (C-Kermit in this case) sends will still start with Ctrl-A.

```
MS-Kermit>connect
C-Kermit>set rec start 2
C-Kermit>server
Alt-X
MS-Kermit>set send start 2
MS-Kermit>send oofa.txt
```

SET {SEND, RECEIVE} TIMEOUT *number*
Timeout interval, in seconds. How long to wait for a packet (SEND), or how long the other Kermit should wait for a packet (RECEIVE), before timing out and trying the same packet again. Example:

```
MS-Kermit>set rec tim 3
```

The SET KEY Command

Terminate the SET KEY command by pressing the Enter key. Kermit will prompt you for a key to be pressed and then for a new definition:

```
MS-Kermit>set key
 Push key to be defined:
 Enter new definition:
```

The definition may be:

- A single character, like `X`.

- A backslash code representing a single character, like `\127`.
- A character string, like `Hello there!`.
- A character string containing backslash codes, like `\7Help!\13`.
- A Kermit verb, like `\kexit`.
- A character string containing any combination of Kermit verbs and backslash codes.
- An empty definition (just press Enter) to remove the key's current definition.
- *Ctrl-C* to cancel the SET KEY command and preserve the key's old definition.
- A question mark to show the kinds of definitions available.

Pressing a key that has been defined this way causes Kermit to send the character(s) that have been assigned to it to the host.

SET KEY CLEAR removes all definitions and restores the built-in default set. SET KEY OFF means use DOS rather than the BIOS (if applicable) to obtain keystrokes; SET KEY ON means use the system BIOS.

A key definition may also be entered on one line by including the key's scan code:

```
SET KEY \315 login\13
```

This assigns the string `login`, followed by a carriage return (`\13`) to the F1 key, whose scan code is `\315` (see Table II-6).

The SET TERMINAL Command

The following commands control many aspects of terminal emulation. To examine their current settings, use the SHOW TERMINAL command.

SET TERMINAL {NONE, VT52, HEATH-19, VT102, VT320, TEK4010}
Selects the type of terminal to emulate. VT320 is the default. Example:

```
MS-Kermit>set term heath
```

SET TERMINAL BELL {AUDIBLE, VISUAL}
Controls whether arriving BEL characters (ASCII character 7) ring the PC's bell[23] (beep) or flash the screen. AUDIBLE is the default.

[23]The noise made by a terminal is called a bell because the earliest terminals, Teletypes (vintage 1930-1970), actually had bells. Modern terminals usually beep.

SET TERMINAL CHARACTER-SET *name*

Selects the terminal character set. Ability to use these character sets depends on which kind of terminal is being emulated. Only the VT320 has international characters. Names of 7-bit character sets are ASCII, BRITISH, DUTCH, FINNISH, FRENCH, FR-CANADIAN, GERMAN, ITALIAN, NORWEGIAN/DANISH, PORTUGUESE, SPANISH, SWEDISH, and SWISS. Names of 8-bit character sets are LATIN1 (the default), DEC-MCS (DEC Multinational Character Set), TRANSPARENT (the current IBM code page), and ALTERNATE-ROM. Examples:

```
MS-Kermit>set term vt320
MS-Kermit>set term char italian
```

SET TERMINAL CLEAR-SCREEN

Clears the screen and screen memory:

```
MS-Kermit>set term clear
```

SET TERMINAL COLOR *number* [, *number* [, *number*]]

Set foreground and background color during terminal emulation:

0	For no snow on IBM Color Graphics Adapter (CGA)
1	High-intensity foreground
10	For fast screen update on IBM EGA
3*x*	Foreground color
4*x*	Background color

where *x* is the sum of any of 1 (Red), 2 (Green), and 4 (Blue). Examples:

```
MS-Kermit>set term color 34, 47
MS-Kermit>set term color 1, 31, 45
```

SET TERMINAL CONTROLS {7-BIT, 8-BIT}

When emulating a VT320, tells whether to send VT320 7-bit or 8-bit control sequences when DEC function keys are pressed. The default is 7-BIT. Example:

```
MS-Kermit>set term cont 8
```

SET TERMINAL CURSOR-STYLE {BLOCK, UNDERLINE}

Selects the cursor style. The default is underline. Example:

```
MS-Kermit>set term curs b
```

SET TERMINAL DIRECTION {LEFT-TO-RIGHT, RIGHT-TO-LEFT}

Chooses the direction in which characters are written on the screen during terminal emulation. Normally left to right. Use right to left for Hebrew, Arabic, or just for fun:

```
MS-Kermit>set term dir right
```

SET TERMINAL GRAPHICS *name*

Specifies the type of graphics adapter in your PC: CGA, EGA, VGA, and others. MS-DOS Kermit will automatically try to figure out what kind of adapter you have. Use this command if it guesses incorrectly. Example:

```
MS-Kermit>set term gr cga
```

SET TERMINAL KEYCLICK {ON, OFF}

On keyboards that support this, turns keyclick on or off. Example:

```
MS-Kermit>set term keyc off
```

SET TERMINAL KEYPAD {APPLICATION, NUMERIC}

Puts the numeric keypad into the specified mode. NUMERIC means send the digits or punctuation marks on the top of the key label; APPLICATION means send the DEC terminal keypad escape sequences associated with the bottom of the key label (arrows, for example). Affects the definition of the keyboard verbs associated with the DEC keypad (see Table II-3). Example:

```
MS-Kermit>set term keyp appl
```

SET TERMINAL MARGIN-BELL {ON, OFF}

Whether to ring the bell (or flash if SET TERM BELL VISIBLE) when the cursor passes column 72 on the screen. Normally OFF, meaning it doesn't make a sound. Example:

```
MS-Kermit>set term marg on
```

SET TERMINAL NEWLINE {ON, OFF}

ON means to send both a carriage return and a linefeed when you press Enter. OFF means to send only a carriage return (this is the default). ON is useful when two PC users are chatting with each other in Kermit CONNECT mode:

```
MS-Kermit>set term newl on
MS-Kermit>set local-echo on
MS-Kermit>connect
```

SET TERMINAL ROLL {ON, OFF}

ON means to restore rolled-back screens to the end when new characters arrive. OFF means to display new characters at the current cursor position, even if it is in a rolled-back screen. OFF is the default. Example:

```
MS-Kermit>set term roll on
```

SET TERMINAL SCREEN-BACKGROUND {NORMAL, REVERSE}

REVERSE changes the foreground color to background, and vice versa. NORMAL is normal. Example:

```
MS-Kermit>set term scr r
```

SET TERMINAL TABSTOPS {AT *n*, CLEAR AT *n*, CLEAR ALL}

Sets or clears screen tab stops at the specified positions. *n* can be a single number, a list of numbers, or *position:interval* to set tabs beginning at the specified *position*, every *interval* spaces, for example, SET TAB AT 1:10. By default, tabs are set every eight spaces. Examples:

```
MS-Kermit>set term tab clear all
MS-Kermit>set term tab at 1, 2, 4, 8, 16, 32, 64
MS-Kermit>set term tab 1:8
```

SET TERMINAL TEK {ENABLE, DISABLE}

Tells MS-DOS Kermit whether it should automatically enter Tektronix graphics mode upon receipt of a special escape sequence from the host. Many popular graphics-oriented host applications send this special sequence. This feature is ENABLEd by default. Example:

```
MS-Kermit>set term tek dis
```

SET TERMINAL WRAP {ON, OFF}

Many host computers will automatically break long lines into a series of lines that fit on your screen. Kermit assumes the host will do this, and so the default is OFF. If your host does not wrap long lines itself, the extra characters will "fall off" the right edge of your screen (or left edge, depending on SET TERM DIRECTION). To fix this:

```
MS-Kermit>set term wrap on
```

SHOW Commands

SHOW COMMUNICATIONS

Communication parameters: Port, speed, parity, flow control, handshake, echo, modem signals:

```
MS-Kermit>show comm

  Communications port: COM1          Speed: 9600
  Local echo: off                    Parity: none (8-bit data)
  Handshake used: none               Flow control: xon/xoff
  Duplex: full                       Display: Regular, 7-bit
  Debug: off

  Modem is ready:      DSR is on
  Carrier Detect:      CD  is on
  no Clear To Send:    CTS is off
```

SHOW FILE

File-related parameters: current path, destination, EOF mode, display, incomplete, warning, take-echo:

```
MS-Kermit>sho file
```

```
  Path: C:\PAM                Discard incomplete file
  File destination: Disk      Warning (filename change): On
  EOF mode: NoCtrl-Z          Take-echo: Off
  Display: Regular, 7-bit     Attribute packets: On
```

SHOW KEY

Displays the definition of a selected key or all defined keys. You are asked to push a key, and Kermit shows you the definition:

```
MS-Kermit>sh key
Push key to be shown (? shows all): (F1 key is pressed)
 Scan Code \315 decimal is defined as
 Verb: Gold  \KGold
 Free space: 128 key & 128 string definitions, 1000 chars
```

SHOW LOGGING

Shows names and status of session, packet, transaction logs, and the screen dump file:

```
MS-Kermit>show log
```

SHOW MACROS [*name*]

Displays the name(s) and definitions of the given macro(s). Example:

```
MS-Kermit>sh mac ibm
 IBM = set timer on<cr>
  set parity mark<cr>
  set local-echo on<cr>
  set handshake xon<cr>
  set flow none<cr>
 Free space (bytes) for names: 993
```

The symbol `<cr>` means carriage return, which is what Kermit substitutes for the comma in your actual macro definition. If you don't give a macro name, Kermit shows all defined macros. To see the definition of a variable, just type its name, for example, `\%1`.

SHOW MEMORY

Displays free memory:

```
MS-Kermit>sho mem
 DOS free memory (bytes): 251,024+48
 Total free bytes: 251,072
```

The first line shows the size of each free piece, and the second line shows the total size of all free pieces.

SHOW MODEM

Displays the status of the Carrier Detect (CD), Data Set Ready (DSR, meaning the modem), and Clear to Send (CTS) modem signals:

```
MS-Kermit>show modem
  Modem is ready:     DSR is on
  Carrier Detect:     CD  is on
  no Clear To Send:   CTS is off
```

SHOW PROTOCOL

Shows the values of the SET SEND, SET RECEIVE, and other file transfer protocol parameters:

```
MS-Kermit>show proto
```

In Kermit's display, ^A means Ctrl-A, ^M means Ctrl-M or carriage return, ^@ means ASCII character 0 (NUL), S: means a SET SEND parameter, and R: means a SET RECEIVE parameter.

SHOW SCRIPTS

Displays values of SET INPUT, SET ALARM, and SET COUNT parameters.

```
MS-Kermit>show scr
  Input echoing: on          Case sensitivity: Ignore
  Timeout (seconds): 1       Timeout-action: Proceed
  Alarm time: 00:00:00       Errorlevel: 0
  INPUT-buffer-length: 128   Take/Macro COUNT: None active
```

SHOW SERVER

Displays server-related parameters, including SET SERVER and ENABLE or DISABLE.

```
MS-Kermit>show serv
  Timeout (sec) waiting for a transaction: 0
  Login Username:
  Server commands available to remote user:
  CD/CWD: enabled        HOST:    enabled
  DELETE: enabled        LOGIN:   enabled
  DIR:    enabled        MESSAGE:enabled
  FINISH: enabled        SPACE:   enabled
  GET:    enabled        TYPE:    enabled
```

SHOW STATISTICS

Displays statistical information on the most recent file transfer and values accumulated since stating Kermit.

```
MS-Kermit>sho stat
```

Kermit's file transfer efficiency is the file characters per second times 10 divided by the baud rate (COM ports only).

SHOW TERMINAL

Displays values of SET TERMINAL parameters:

```
MS-Kermit>sho term
```

The bottom line shows the tab settings: T for each tab stop. The numbers mark every ten spaces.

SHOW TRANSLATION

Lists codes for bytes received during CONNECT and their new values for screen display as established by the SET TRANSLATE INPUT command, plus whether the translation mechanism is ON or OFF (default is OFF).

```
MS-Kermit>set transl in \91 \123
MS-Kermit>set transl in \93 \125
MS-Kermit>sho on
MS-Kermit>sho transl
 Input Translation is on
 Translation table of rec'd bytes while in CONNECT mode -
 Format: [received byte (decimal) -> local byte (decimal)]
 [\91 -> \123]  [\93 -> \125]
```

Glossary

Acoustic Coupler
A modem having two rubber cups into which the telephone handset is inserted.

Alt
The IBM PC key that you hold down while pressing another key in order to produce an Alt character. For example, `Alt-X` is produced by holding down Alt and pressing X.

Analog
Representation of computer data in some other form, like the kind of sound waves that are transmitted over telephone lines by modems. *Also see* Digital.

Answer
One of two modes a modem can be in. In answer mode, the modem waits for a call. *Also see* Originate.

ASCII
American Standard Code for Information Interchange. A 128-character code widely used by computers for representing and transmitting character data, in which each character corresponds to a number between 0 and 127. The ASCII alphabet is listed in Table II-4.

Asynchronous
Character- or byte-oriented data transmission, with delimitation of characters accomplished by start and stop bits. Used by PC serial ports and modems.

Asynchronous Adapter
The PC circuit board that controls the serial communication port, the device used by Kermit for connections to external modems and for direct connections to other computers. There are several kinds of asynchronous adapter. PC and PC/XT computers have a half-height card, called the Asynchronous Adapter, with one 25-pin male connector. PC/ATs may also use this card, or they may have the PC/AT Serial/Parallel Adapter with two connectors: a 9-pin male serial connector (the communication port) and a 25-pin female parallel connector (the printer port, which should not be used for communications). The PS/2 has a built-in 25-pin male connector for communications, not to be confused with the 25-pin female connector, which is the parallel printer port. Additional ports may be installed in the PS/2 using the Personal System/2 Dual Async Adapter/A, which has two male 9-pin connectors.

Autoanswer
A kind of modem that automatically answers a telephone call without manual intervention.

Autodial
A kind of modem that simulates a telephone's dialing mechanism, rotary or Touch-tone, in order to place a call, usually under computer control.

Baud
As most commonly used in PC communications, "baud" is the transmission speed, expressed in bits per second (bps).

Binary
Referring to the number two. Binary notation is a way of writing numbers using only the two digits 0 and 1. Computers are made out of switches that have only two states, on (1) and off (0).

Binary File
A file containing codes that are used to control a device like a computer or printer. The contents of binary files usually depend on some particular hardware, and they should not be converted or translated in any way during transfer to another system.

BIOS
IBM's Basic Input Output System. The part of DOS that controls devices such as the disk, keyboard, serial port, and screen.

Bit
A binary digit, 0 or 1.

Block Check
A quantity formed from a block (packet) of data by combining all its bytes, and then transmitting the result with the block itself so that the receiver of the block can determine whether it was corrupted in transit. Kermit supports three types of block checks: a one-character checksum (6 bits), a two-character checksum (12 bits), and a three-character CRC (16 bits). *Also see* Checksum, CRC.

bps
Bits per second. Usually equivalent to baud.

BREAK
A binary zero on a communication line lasting about 0.275 seconds and generated by pressing `Alt-B` or `Ctrl-]B` on the PC keyboard during Kermit terminal emulation. Also, a "long BREAK" lasting about 1.5 seconds, produced by typing `Ctrl-]L`.

Buffer
A place to put arriving data until the intended recipient can get around to reading it, or a place to store outbound data until the transmitter gets around to sending it.

Byte
A unit of storage, abbreviated B, intended to hold a character, usually 8 bits long. Computer memory and disk capacity are often measured in thousands (K) or millions (M) of bytes (for example, 256KB).

Carriage Return
ASCII character number 13. This is the character that is transmitted when you press the PC's Enter key and that is used in conjunction with linefeed to terminate lines of text in IBM PC text files. Abbreviated CR.

Carrier
A continuous signal that is sent between two modems. The presence of carrier tells one modem that the other modem is in data transmission mode. The loss of carrier indicates the data connection is broken. An external modem usually has a carrier status light to let you know that it is communicating with the other modem.

CD
Carrier Detect. A signal to the PC from the modem indicating that it is connected to another modem. *Also see* Carrier.

CGA
The IBM PC Color Graphics Adapter.

Character
A discrete unit of textual or control information, such as a letter, digit, or punctuation mark, belonging to a particular character set, like ASCII, IBM Code Page 437, or ISO Latin Alphabet 1.

Checksum
A block check based on the arithmetic sum of all the bytes in a block.

Circuit Board
A flat rectangular board containing electronic circuits that implement some component of a computer or communication device. Designed to be plugged into a slot, with signals passing through contacts on its edge. The asynchronous adapter is a circuit board, and so is an internal modem.

Code
In data communications, the numeric or internal representation for a character in a particular character set, like ASCII or IBM Code Page 437.

Code Page
The name IBM uses for its character sets (see Table II-5).

Communication Port
A device that allows a computer or terminal to engage in data communication, appearing as an external connector on the back of a PC for a cable to a modem or another computer. *Also see* Asynchronous Adapter.

Connector
A plug, of either male or female gender, that provides contacts for one or more wires within a cable and that mates with a similar plug of opposite gender to provide the desired electrical circuits. The connectors used most commonly with PCs for data communication are D-connectors (so called because they are shaped like the letter D) with either 25 pins (DB-25) or 9 pins (D-9). The DB-25 is often called an RS-232 or EIA connector.

Console
The primary input/output device with which a person controls a personal computer or a time-sharing session on a shared computer. The keyboard and screen.

Control Character
An ASCII chararacter in the range 0 to 31, or ASCII character 127, contrasted with the printable, or graphic, characters in the range 32 to 126 (see Table II-4). Produced on an ASCII terminal by holding down the Ctrl key and typing the desired character. Standard 8-bit character sets such as ISO Latin-1 also have 32 additional control characters in the range 128 to 159.

CPU
Central processing unit. The part of the computer that executes instructions, together with its memory, distinct from external devices (peripherals).

CR
Carriage return (ASCII 13, Control-M).

CRC
Cyclic redundancy check. An error-checking technique in which a block of data is viewed as a long sequence of bits, to be divided by a certain binary number, with the remainder used as the block check. *Also see* Block Check.

CRLF
Carriage return and linefeed, the sequence of ASCII characters (numbers 13 and 10) used by MS-DOS to delimit lines in a text file.

CTERM
The DECnet virtual terminal protocol used by the VAX/VMS SET HOST command.

Ctrl
Control. The key that you hold down while pressing another key (a letter or certain punctuation marks) in order to produce a control character. For example, `Ctrl-C` is produced by holding down Ctrl and pressing C.

CTS
Clear To Send, the modem signal that indicates readiness to accept data.

Cursor
The rectangular block or underscore on your CRT screen that indicates the current position.

Data
Information as it is stored in, or transmitted by, a computer or terminal.

Dead Key
On certain IBM PC national keyboards, a key that acts as a prefix for another key, to produce a special character. Press the dead key first, then the letter. For example, on the German keyboard, apostrophe followed by e produces e-acute; apostrophe is the dead key.

Dedicated Line
A communication line that connects two devices with relative permanence, for example, a direct line from a terminal to a computer, or a leased telephone circuit. The opposite of a switched or dialup line.

Default
The value that is used for some parameter when no other value is explicitly provided. For example, the default Kermit block check is 1, and it will be used unless you explicitly tell Kermit to use type 2 or 3 by using the SET BLOCK-CHECK command. *Also see* Block Check.

Dialup
A data connection established with a telephone call, usually involving modems.

Digital
Representation of data by discrete 0s and 1s rather than continuous (analog) voltages. A PC is digital internally, and its communication port is digital. A modem converts digital computer data to analog waveforms (similar to speech) for transmission on telephone lines.

Directory
A file on a disk that contains a list of other files, with their physical locations on the disk, and possibly other information about them, such as size and creation date.

Directory Name
In MS-DOS, a sequence of characters that identifies a particular directory. The directory name is followed by a backslash character (\), for example, `PROGRAMS\`. Each disk can have many directories, and each directory can contain other directories. *Also see* Subdirectory.

Disk
A rotating magnetic storage medium for digital information, similar to a phonograph record, but possibly having more than one platter mounted on a central spindle. Disks are generally classified as hard (usually permanent, high capacity) and floppy (single platter, flexible, removable, moderate capacity). Floppy disks are also called diskettes.

Diskette
A single-platter, removable disk. Can be flexible, like an 8-inch or 5.5 inch floppy diskette, or rigid, like the 3.5-inch diskette used in the IBM PS/2.

DOS
Disk Operating System. A computer operating system that uses a magnetic disk as its principal medium of permanent storage. Also, short for MS-DOS and PC-DOS.

Download
To transfer a file from another computer to your PC.

DSR
Data Set Ready. A signal from a modem to the PC that says the modem is turned on and in data mode.

DTR
Data Terminal Ready. A signal from the PC to a modem that says the PC is turned on and ready to communicate. Some modems will refuse to communicate with your PC unless the PC is sending the DTR signal. MS-DOS Kermit starts sending the DTR signal as soon as you give the CONNECT, PAUSE, or WAIT command and keeps sending it until you give the HANGUP command.

Duplex
The degree to which a channel permits two-way traffic. Half duplex means traffic can go either way, but only one way at a time; full duplex means traffic can go both ways at the same time. *Also see* Echo.

EBCDIC
Extended Binary Coded Decimal Interchange Code. The character code used on IBM mainframes. Not covered by any formal standards but described definitively in the IBM System/370 Reference Summary.

Echo
How a character typed at a terminal, or a device emulating a terminal, is sent to the screen. Local echo means the terminal itself copies the character to the screen; usually associated with half-duplex communication. Remote echo means the system to which the character is transmitted sends it back to be displayed, possibly modified; this can be done only on full-duplex connections.

EGA
The IBM PC Enhanced Graphics Adapter.

Enter
The IBM PC key that terminates a command or a line of text. During terminal emulation, the Enter key sends a carriage return, ASCII character 13.

ESC
ASCII character 27, Control-[.

Escape Character
A character used to get the attention of Kermit during CONNECT mode. Not to be confused with ASCII ESC. MS-DOS Kermit's default escape character is `Ctrl-]`.

Escape Sequence
A sequence of characters that selects a certain function. For instance, Kermit, during CONNECT, will accept a variety of escape sequences as commands. These consist of Kermit's escape character followed by a single character that selects the function, for example, `Ctrl-]B` to send a BREAK signal.

Ethernet
A local area network technology.

Even
See Parity.

External Modem
A modem that is not mounted internally in a PC. Usually portable, requiring its own power source or drawing power from the telephone line.

File
A collection of data that is stored on a disk and that has a name.

File Group
A collection of files that can be referred to using a single file specification that contains wildcard characters. *Also see* Wildcard.

Filename
In MS-DOS, a sequence of one to eight characters, followed by a period (`.`), followed by zero to three characters, used to identify a file within a directory, for example, `OOFA.TXT`.

File Specification
In MS-DOS, a sequence of characters composed of a disk letter followed by a colon (`:`), followed by a directory name enclosed in backslashes (`\`), followed by a filename, for example, `A:\PROGRAMS\OOFA.C`. If the disk is the same as the current disk, the disk designator can be omitted, for example, `\PROGRAMS\OOFA.C`. If the disk and directory are the current ones, both disk and directory designators can be omitted, for example, `OOFA.C`. If the disk is different but the directory is the same, the directory can be omitted, for example, `A:OOFA.C`. The filename portion of a file specification can include the wildcard characters `*` and `?` to denote a file group. *Also see* Wildcard.

Flow Control
The process by which the flow of data in a particular direction is regulated so that the arrival of data is coordinated with the capacity of the receiver to process it. MS-DOS Kermit uses Xon/Xoff flow control by default. If Kermit's input buffer is in danger of filling up, it sends an Xoff (Ctrl-S) character. When it is ready to receive more data, it sends an Xon (Ctrl-Q). If it receives an Xoff, it stops sending; when it receives an Xon, it resumes sending.

Front End
A communication processor for a host computer, which operates independently from it but is closely tied to it. The front end relieves the host from the burden of detailed control of multiple terminals.

Full Duplex
A channel that permits simultaneous two-way data traffic between two devices, usually by dedicating one wire to each direction. Full-duplex connections usually use Xon/Xoff flow control and remote echo. *Also see* Flow Control, Echo.

Half Duplex
A channel that permits data transmission in both directions, but only in one direction at a time. Half-duplex connections usually use line-turnaround handshake and local echo.

Handshake
A method for granting permission to transmit a half-duplex channel. Usually the terminal or PC sends a carriage return as its handshake, and the host uses the Xon character (Ctrl-Q).

Hex
Slang for hexadecimal.

Hexadecimal
Numeric notation in base 16, using the digits 0–9 and A–F to represent the numbers 0–15. For example, 9 hex is 9 decimal, the hex A is 10 decimal, F hex is 15 decimal, and 10 hex is 16 decimal.

Host
A multiuser computer or time-sharing system to which a terminal (or a PC emulating a terminal) may be connected.

Input/Output
The process of getting data into and out of a computer, whether from a peripheral device like a disk or through a communication line to a terminal or another computer. Called I/O for short.

Interface
Computer jargon for something that allows two otherwise incompatible components to work together by satisfying their respective physical and logical requirements and making any necessary conversions of format, timing, voltage, etc. A connector is a kind of interface; so is the serial port. The aspect of a software program that interacts with a person is sometimes called the user interface. The console is said to be the user's interface to the system.

I/O
Input/Output.

ISO
International Organization for Standardization. A voluntary international group of national standards organizations that issues standards in all areas, including computers and information processing.

ISO Standard 8859
An ISO standard specifying a series of 8-bit computer character sets that include characters from many languages. These include the ISO Latin Alphabets 1–5, which cover most of the written languages based on Roman letters, plus special character sets for Cyrillic, Greek, Arabic, and Hebrew.

K
Abbreviation for kilo, meaning either 1000 or 1024.

LAN
Local area network.

LAT
Local Area Transport protocol, used by DEC Ethernet terminal servers.

Latin Alphabet
See ISO Standard 8859.

Leased Line
A permanent, dedicated communication line rented from the telephone company or another company.

Line
(1) A physical communication path, such as a telephone cable. (2) A sequence of characters in a text file intended to print on one line of a page or screen.

Linefeed
ASCII character 10. Used in conjunction with carriage return (ASCII 13) to delimit lines of text in an MS-DOS text file.

Local
Nearby, close to. When two systems or devices are connected, the local system is the closer one. When two Kermit programs are connected, the local Kermit is the one that the user interacts with most directly (the one that has the CONNECT command).

Local Area Network
A data communication network that allows computing devices in a building or on a campus to communicate at high speeds.

Local Echo
Immediate display on the local screen, by the terminal or PC, of characters sent to a remote computer. Associated with half-duplex communication.

Loopback Connector
A data connector that sends back to the computer whatever it receives.

M
Abbreviation for mega, meaning either one million or 1,048,576.

Mainframe
Commonly used to mean a big computer, as distinct from a minicomputer or microcomputer. In this book, it means any multiuser computer in which a user's console is also the user's only communication channel with the computer.

Mark
See Parity.

Medium
Something through which data is transmitted—copper wire, coaxial cable, optical fiber, empty space—or on which it is stored—magnetic disk, diskette, tape, CD ROM.

Memory
The internal, volatile, high-speed, solid-state storage of a computer, as distinguished from external, permanent, lower speed, rotating mechanical memories (for example disks, tapes) used for bulk storage.

Message
A unit of information, usually consisting of multiple bytes or characters, put into some specified format for transmission.

Microcomputer
In this book, any single-user computer whose console is distinct from its communication line.

Millisecond
One thousandth of a second.

Modem
Modulator/demodulator. A device that converts between serial digital data as output from a computer and analog waveforms suitable for transmission on a telephone line.

Modem Eliminator
See Null Modem.

Modem Signals
Signals transmitted from a modem to the PC, or vice versa, by which the modem and PC tell each other their status. The modem gives the CD (Carrier Detect), DSR (Data Set Ready), and CTS (Clear To Send) signals to the PC, and the PC gives the DTR (Data Terminal Ready) and RTS (Request To Send) signals to the modem. *Also see* CD, CTS, DSR, DTR, and RTS.

MS-DOS
Microsoft's Disk Operating System for microcomputers based on the Intel 8086 family of CPU chips. *Also see* PC-DOS.

Multiplexer
A device that allows multiple devices to share a single communication medium. Used in pairs, one at each end; the transmitter multiplexes, the receiver demultiplexes.

Network
A permanent arrangement that allows two or more computers or devices to communicate with each other conveniently and reliably at high speeds, over dedicated media, and that typically requires special hardware and operating-system level software.

Noise
Corruption of data during transmission.

NUL
ASCII character number 0, as distinct from the number zero or the ASCII character digit 0 (ASCII 48). Also, the MS-DOS null device.

Null Modem
A pair of connectors, possibly with a length of cable between them, that allows two computers or terminals to be directly connected without intervening modems or multiplexers, and that supplies the required modem signals by means of cross-connections and jumpers.

Odd
See Parity.

Off
(1) Not in effect (said of an option). (2) Zero (said of a bit).

On
(1) In effect (said of an option). (2) One (said of a bit).

Operating System
The software program that controls a computer at its most basic level.

Originate
The mode of operation for a modem when it places a data call, as opposed to receiving one.

OS
Operating system.

OS/2
Operating System/2, a possible successor to MS-DOS and PC-DOS for the IBM PS/2 line of computers and compatibles.

Packet
A message that consists of fields whose locations and interpretation are agreed upon by the sending and receiving entities, to be transmitted as a whole, and that typically contains sequencing, error checking, and other control information as well as data.

Parallel Port
A PC device used for connecting a parallel printer and that appears as a 25-hole female connector on the back of the PC. It should not be used for communication with modems or other computers.

Parameter
A symbolic value standing for, or to be replaced by, a real value. For example, in LOG SESSION *filename*, *filename* is a parameter to be replaced by the name of an actual file.

Parity
An error detection method in which one bit in each 8-bit byte is set aside to indicate some property of the remaining bits in a byte or word. Odd parity means the parity bit is set to make the overall number of 1 bits odd; even makes the overall number of 1 bits even. Mark parity means the parity bit is always set to 1; space parity means it's always set to zero. No parity means the bit that would otherwise be used for parity may be used for data, and 8-bit data may be transmitted.

PBX
Private branch exchange. A telephone system that serves the internal needs of an organization and and that provides connections to the external phone system. Often used for data transmission as well as voice within the organization. May be digital or analog.

PC
Personal computer. In this book, the term PC refers to the entire IBM PC and PS/2 families and compatibles. *Also see* Microcomputer.

PC-DOS
The version of MS-DOS distributed by IBM for use on its PC and PS/2 familes. *Also see* MS-DOS.

Port
See Communication Port.

Port Contention Unit
A device that allows multiple terminals to be connected to multiple computers, in which terminal ports contend for computer ports. Typically, the port contention unit engages in a dialog with the user, asking which computer the user wishes to connect to.

Protocol
In data communication, a set of rules and formats for exchanging messages, generally incorporating methods of sequencing, timing, and error detection and correction.

Public Data Network
A network providing access, on a subscription basis, to widely scattered and diverse services. Telenet, Tymnet, and Datapac are examples.

Relative Directory
In MS-DOS, a directory name that does not start with a slash and is assumed to be a subdirectory of the current directory.

Remote
Said of the more distant, or less directly accessed, of two connected devices. A remote Kermit is the one running on the host that the local Kermit has connected to.

Retry
In this book, a second or subsequent attempt at transmitting a particular Kermit packet.

ROM
Read-only memory. High-speed internal memory containing permanently recorded information.

RS-232-C
An Electronic Industries Association (EIA) standard that gives the electrical and functional specification for serial binary digital data transmission. The most commonly used interface between terminals (or computers) and modems (or multiplexers).

RTS
Request To Send. A signal used by a terminal or computer to ask permission of a modem to transmit data to it.

RTS/CTS
A form of flow control that uses the RTS and CTS modem signals. When the PC has characters to send, it turns on the Request To Send (RTS) signal. When the host is ready to receive the PC's characters, it turns on the Clear to Send (CTS) signal. When the PC is finished transmitting, it turns off RTS to give permission to the host to transmit.

Serial
In series, sequential, one after another. The dominant mode of transmission of binary data over distances greater than a few feet.

Serial Port
See Asynchronous Adapter.

Server
A program or intelligent device that provides specified services to users, or clients, in response to requests, usually over a communication line or network. Kermit programs can be put into server mode, in which they accept commands only from other Kermit programs.

Space
See Parity.

Subdirectory
In MS-DOS, a directory within another directory. In a file specification, each subdirectory name is preceded by a backslash (\) character, for example:

```
C:\PROGRAMS\SOURCE\OOFA.C
```

Switched Line
A communication line subject to switching, like a dialed telephone connection.

Telecommunication
Asynchronous serial data communication, possibly (but not necessarily) involving dialup telephone connections and modems.

Telenet
One of the subscriber-based public packet switched networks in the United States.

Terminal
A device that allows a person to interact with a computer, with the person typing characters on a keyboard to send them to the computer, and with the computer's responses appearing on a screen or printer. Sometimes includes the ability to interpret special character sequences to accomplish screen formatting. In general, differs from a computer by not having local permanent memory or general-purpose programmability.

Terminal Emulation
Behaving like a terminal. Said of software that runs on PCs or other computers that sends the user's keystrokes out the serial port and sends the port input to the screen. Sometimes includes the ability to interpret the same special sequences that a specific real terminal would obey. The Kermit CONNECT command performs terminal emulation.

Terminal Server
A network device that allows ordinary terminals with no networking capabilities of their own to participate in a network, provided hosts share a common protocol with the terminal server.

Text
Computer data intended for a person to read, or typed by a person, that consists of only printable characters and those control characters necessary for format control (carriage return, linefeed, tab, formfeed). Text files can be transferred between unlike systems and still remain useful. *Also see* Binary File.

Timeout
The process by which a program wakes up after waiting for some expected event (like input from a device) longer than a specified amount of time.

Translation Table
A list of all the translations from one character set into another.

Tymnet
A public packet switched network service offered by Tymnet, Inc.

Unattended
Referring to an operation that can proceed automatically without human intervention.

UNIX
A popular operating system developed at AT&T Bell Laboratories and noted for its portability.

Upload
To transfer a file from your PC to another computer.

User
A person who is using a computer.

User Interface
The hardware and software with which a person communicates with a computer.

Wildcard
A notation for referring to a group of files in a single file specification, by including pattern-matching characters. In MS-DOS, a * character in a filename matches any sequence of characters, and a ? character matches any single character.

Workstation
A single-user computer. Equivalent to a PC or microcomputer in that the console is separate from the communication line but is usually composed of more expensive components. Intended for more ambitious uses.

Xon/Xoff
The most common full-duplex flow control method, in which the receiver sends an Xoff character when its input buffer is close to filling up and an Xon when it has made room for more data to arrive.

Appendix I

The MS-DOS Kermit Distribution Diskette

`READ.ME`
Explanation of the files on the diskette. Be sure to read this file (use the DOS TYPE command) in case the diskette has been updated since the publication of this book.

`KERMIT.HLP`
A brief help file listing the MS-DOS Kermit commands.

`KERMIT.EXE`
The MS-DOS Kermit program, ready to run.

`MSKERMIT.INI`
Sample initialization file for MS-DOS Kermit. Includes many of the macro definitions from Chapter 14. Edit this file to suit your needs and preferences.

`WP.INI`
An initialization file for using MS-DOS Kermit with host-resident versions of WordPerfect.

`EMACS.INI`
Key settings for using the EMACS editor.

`KERMIT.UPD`
A list of features that are new to the current release of MS-DOS Kermit.

`KERMIT.BWR`
A list of known problems and limitations of the current release of MS-DOS Kermit.

KERMIT.PIF

Program Information File for using MS-DOS Kermit under Microsoft Windows. MS-Windows users should copy this file into the area where their other .PIF files are stored.

XSEND.EXE

A program for building MS-DOS Kermit command files for sending entire directory trees from one PC to another, for example, to back up or copy a hard disk.

XSEND.HLP

Instructions for using XSEND.

XSEND.C

C-language source code for XSEND.

EPSON.COM

A printer driver that allows Kermit's Tektronix Graphics screens that are displayed by an EGA monitor to be printed on an Epson printer.

EPSON.ASM

Assembly-language source program for EPSON.COM.

EPSON.HLP

Instructions for using EPSON.

CHARDEMO.VT

A demonstration of MS-DOS Kermit's character sets. To view this demonstration, start MS-DOS Kermit, then give the commands:

```
MS-Kermit>set terminal vt320
MS-Kermit>replay chardemo.vt
```

FEATURES.VT

A demonstration of some of the features of VT-series terminals and how MS-DOS Kermit emulates them. To view this demonstration, start MS-DOS Kermit, then give the commands:

```
MS-Kermit>set terminal vt320
MS-Kermit>replay features.vt
```

CASTLE.HGR

A demonstration of some of the Heath/Zenith-19 terminal emulation features of MS-DOS Kermit. To view this demonstration, start MS-DOS Kermit, then give the commands:

```
MS-Kermit>set terminal heath
MS-Kermit>replay castle.hgr
```

You should see a picture of a castle.

PATTERN.HGR

Another demonstration of some of the Heath/Zenith-19 terminal emulation features of MS-DOS Kermit. To view this demonstration, start MS-DOS Kermit, then give the commands:

```
MS-Kermit>set terminal heath
MS-Kermit>replay pattern.hgr
```

DEMO.TEK

A demonstration of some of MS-DOS Kermit's Tektronix emulation features. To view this demonstration, start MS-DOS Kermit, then give the commands:

```
MS-Kermit>set terminal tek
MS-Kermit>replay demo.tek
```

USA.TEK

A demonstration of Kermit's Tektronix 4010 graphics capability. To view this demonstration, start MS-DOS Kermit, then give the commands:

```
MS-Kermit>set terminal tek
MS-Kermit>replay usa.tek
```

HAYES.TAK

TAKE file to be used with the DIAL macro and Hayes modems.

LK250.COM

Driver for using DEC LK250 keyboards on IBM computers.

LK250.ASM

Assembly-language source for LK250 driver.

LK250.HLP

Instructions for using LK250.

Appendix II

Tables

Table II-1 shows the assignments of modem signals to pins for IBM PC 25-pin (DB25) connectors and IBM PC/AT and PS/2 9-pin (DB9) connectors.

Table II-1 RS-232-C Modem Signals and Pins

Signal	*DB25*	*DB9*	*Description*
FG	1	–	Frame (protective) ground
TD	2	3	Transmitted data (from PC to modem)
RD	3	2	Received data (by PC from modem)
RTS	4	7	Request To Send (by PC)
CTS	5	8	Clear To Send (by modem)
DSR	6	6	Dataset Ready (modem is turned on)
SG	7	5	Signal Ground
CD	8	1	Carrier Detect (modems are communicating)
DTR	20	4	Data Terminal Ready (PC is online)
RI	22	9	Ring Indicate (modem tells PC phone is ringing)

MS-DOS Kermit CONNECT-Mode Escapes

Table II-2 lists the keyboard escape sequences available during terminal emulation. The *Character* column shows the key that you press after typing `Ctrl-]`; for example, `C` stands for `Ctrl-]C`. The *Verb* column shows the Kermit verb associated with this key, which you can use with the SET KEY command to assign this function to other keys. A complete list of keyboard verbs is given in Table II-3.

Table II-2 MS-DOS Kermit CONNECT-Mode Escapes

Character	*Verb*	*Description*
`?`	`\Khelp`	Help—prints the available single-character commands
`0`	`\Knull`	(the digit zero) Transmit a NUL (ASCII 0)
`B`	`\Kbreak`	Transmit a BREAK signal
`C`	`\Kexit`	Close the connection and return to `MS-Kermit>` prompt level
`F`	`\Kdump`	File the current screen in the screen dump file
`H`	`\Khangup`	Hang up the phone (or network connection)
`L`	`\Klbreak`	Transmit a long BREAK (1.8 seconds)
`M`	`\Kmodeline`	Toggle the mode line (i.e., turn it off if it is on and vice versa)
`P`	`\Kdos`	Push to DOS; get back to CONNECT mode by typing `exit`
`Q`	`\Klogoff`	Temporarily quit logging the remote session
`R`	`\Klogon`	Resume logging the remote session
`S`	`\Kstatus`	Show the status of the connection
`Ctrl-]`	*none*	(or whatever you have set the escape character to be) Type the escape character twice to send it once to the host

MS-DOS Kermit Keyboard Verbs

Table II-3 shows the assignments of Kermit verbs to keys for use during terminal emulation. To change the assignments, use the SET KEY command, for example:

```
MS-Kermit>set key \324 \Kexit
```

This assigns the EXIT function to the F10 key (see Table II-6 for keyboard scan codes).

Table II-3 MS-DOS Kermit Keyboard Verbs

Verb	*Meaning*	*Assignment*
\Kbreak	Send a BREAK signal	*Alt-B, Ctrl-Break*
\KdecDo	DEC Do key	*none*
\KdecF6	DEC F6 key	*none*
\KdecF7	DEC F7 key (etc., up to 14)	*none*
\KdecF17	DEC F17 key (etc., up to 20)	*none*
\KdecFind	DEC Find key	*none*
\KdecHelp	DEC Help key	*none*
\KdecInsert	DEC Insert key	*none*
\KdecNext	DEC Next Screen key	*none*
\KdecPrev	DEC Prev Screen key	*none*
\KdecRemove	DEC Remove key	*none*
\KdecSelect	DEC Select key	*none*
\Kdnarr	Transmit what DEC Down-Arrow key sends	*Down-Arrow*
\Kdnone	Roll screen down one line	*Ctrl-Page-Down*
\Kdnscn	Roll down (forward) to next screen	*Page Down*
\Kdos	"Push" to DOS	*Ctrl-]P*
\Kdump	Append current screen to dump file	*Ctrl-End*
\Kendscn	Roll down to end of screen memory	*End*
\Kexit	Escape back from CONNECT mode	*Alt-X*
\Kgold	Transmit what DEC Gold key sends	*F1*
\Khangup	Drop DTR so modem will hang up phone	*none*
\Khelp	Display CONNECT help message	*Alt-H*
\Kholdscrn	Toggle hold screen mode	*none*

Table II-3 MS-DOS Kermit Keyboard Verbs (continued)

Verb	*Meaning*	*Assignment*
\Khomscn	Roll up to top of screen memory	*Home*
\Kkp0	DEC keypad 0	*Shift-F7*
\Kkp1	DEC keypad 1	*Shift-F3*
\Kkp2	DEC keypad 2	*Shift-F4*
\Kkp3	DEC keypad 3	*Shift-F5*
\Kkp4	DEC keypad 4	*F9*
\Kkp5	DEC keypad 5	*F10*
\Kkp6	DEC keypad 6	*Shift-F1*
\Kkp7	DEC keypad 7	*F5*
\Kkp8	DEC keypad 8	*F6*
\Kkp9	DEC keypad 9	*F7*
\Kkpcoma	DEC keypad comma	*Shift-F2*
\Kkpdot	DEC keypad dot (period)	*Shift-F8*
\Kkpenter	DEC keypad Enter	*Shift-F6*
\Kkpminus	DEC keypad minus	*F8*
\Klbreak	Send a "long BREAK" signal	*Ctrl-]L*
\Klfarr	Transmit what DEC Left-Arrow key sends	*Left-Arrow*
\Klogoff	Turn off session logging	*Ctrl-]Q*
\Klogon	Turn on session logging	*Ctrl-]R*
\Kmodeline	Toggle modeline off/on	*Keypad minus*
\Knull	Send a null (ASCII 0)	*Ctrl-]0*
\Kpf1	PF1, same as Gold	*F1*
\Kpf2	DEC PF2 key	*F2*
\Kpf3	DEC PF3 key	*F3*
\Kpf4	DEC PF4 key	*F4*
\Kprtscn	Print the current screen	*Print Screen*
\Kreset	Reset terminal emulator to initial state	*Alt-=*
\Krtarr	Transmit what DEC Right-Arrow key sends	*Right-Arrow*
\Kstatus	Display STATUS message	*Alt-S*
\Kterminalr	Invoke user-defined macro TERMINALR	*none*

ASCII Character Codes

Key: *Dec* = decimal value, *Hex* = hexadecimal value, ^X = Ctrl-X.

Table II-4 ASCII Character Codes

Dec Hex	*Name*	*Char*	*Dec Hex*	*Char*	*Dec Hex*	*Char*	*Dec Hex*	*Char*
000 00	NUL	^@	032 20	SP	064 40	@	096 60	‘
001 01	SOH	^A	033 21	!	065 41	A	097 61	a
002 02	STX	^B	034 22	"	066 42	B	098 62	b
003 03	ETX	^C	035 23	#	067 43	C	099 63	c
004 04	EOT	^D	036 24	$	068 44	D	100 64	d
005 05	ENQ	^E	037 25	%	069 45	E	101 65	e
006 06	ACK	^F	038 26	&	070 46	F	102 66	f
007 07	BEL	^G	039 27	’	071 47	G	103 67	g
008 08	BS	^H	040 28	(	072 48	H	104 68	h
009 09	HT	^I	041 29	)	073 49	I	105 69	i
010 0A	LF	^J	042 2A	*	074 4A	J	106 6A	j
011 0B	VT	^K	043 2B	+	075 4B	K	107 6B	k
012 0C	FF	^L	044 2C	,	076 4C	L	108 6C	l
013 0D	CR	^M	045 2D	-	077 4D	M	109 6D	m
014 0E	SO	^N	046 2E	.	078 4E	N	110 6E	n
015 0F	SI	^O	047 2F	/	079 4F	O	111 6F	o
016 10	DLE	^P	048 30	0	080 50	P	112 70	p
017 11	CD1	^Q	049 31	1	081 51	Q	113 71	q
018 12	DC2	^R	050 32	2	082 52	R	114 72	r
019 13	DC3	^S	051 33	3	083 53	S	115 73	s
020 14	DC4	^T	052 34	4	084 54	T	116 74	t
021 15	NAK	^U	053 35	5	085 55	U	117 75	u
022 16	SYN	^V	054 36	6	086 56	V	118 76	v
023 17	ETB	^W	055 37	7	087 57	W	119 77	w
024 18	CAN	^X	056 38	8	088 58	X	120 78	x
025 19	EM	^Y	057 39	9	089 59	Y	121 79	y
026 1A	SUB	^Z	058 3A	:	090 5A	Z	122 7A	z
027 1B	ESC	^[	059 3B	;	091 5B	[	123 7B	{
028 1C	FS	^\	060 3C	<	092 5C	\	124 7C	|
029 1D	GS	^]	061 3D	=	093 5D	]	125 7D	}
030 1E	RS	^^	062 3E	>	094 5E	^	126 7E	~
031 1F	US	^_	063 3F	?	095 5F	_	127 7F	DEL

IBM PC and PS/2 Code Pages and ISO Latin Alphabet 1

Table II-5 shows the IBM PC and PS/2 code pages, listed in alphabetical order for easy reference. To enter these characters, look up the desired character in your current code page, hold down the Alt key and type the three-digit *decimal* number (listed in the *Dec* column) on the PC's numeric keypad, or use the PC's "dead key" combinations listed in the back of the DOS manual (DOS 3.30 or later) with your KEYB driver. Only the special alphabetic and punctuation characters are listed. For ASCII characters in the range 0–127, see Table II-4. CP437 is the original IBM PC code page. CP850 is the multilingual code page. CP860 is for Portugal. CP863 is for French Canada. CP865 is for Norway. The Latin-1 column shows ISO Latin Alphabet 1.

Table II-5 IBM PC and PS/2 Code Pages

Code Page *Character*	**CP437** *Dec Hex*	**CP850** *Dec Hex*	**CP860** *Dec Hex*	**CP863** *Dec Hex*	**CP865** *Dec Hex*	**Latin-1** *Dec Hex*
a-acute	160 A0	160 A0	160 A0		160 A0	225 E1
A-acute		181 B5	134 86			193 C1
a-circumflex	131 83	131 83	131 83	131 83	131 83	226 E2
A-circumflex		182 B6	143 8F	132 84		194 C2
ae digraph	145 91	145 91			145 91	230 E6
AE digraph	146 92	146 92			146 92	198 C6
a-grave	133 85	133 85	133 85	133 85	133 85	224 E0
A-grave		183 B7	145 91	142 8E		192 C0
a-ring	134 86	134 86			134 86	229 E5
A-ring	143 8F	143 8F			143 8F	197 C5
a-tilde		198 C6	132 84			227 E3
A-tilde		199 C7	142 8E			195 C3
a-umlaut	132 84	132 84			132 84	228 E4
A-umlaut	142 8E	142 8E			142 8E	196 C4
c-cedilla	135 87	135 87	135 87	135 87	135 87	231 E7
C-cedilla	128 80	128 80	128 80	128 80	128 80	199 C7
e-acute	130 82	130 82	130 82	130 82	130 82	233 E9
E-acute	144 90	144 90	144 90	144 90	144 90	201 C9

Table II-5 IBM PC and PS/2 Code Pages (continued)

Code Page *Character*	**CP437** *Dec Hex*	**CP850** *Dec Hex*	**CP860** *Dec Hex*	**CP863** *Dec Hex*	**CP865** *Dec Hex*	**Latin-1** *Dec Hex*
e-circumflex	136 88	136 88	136 88	136 88	136 88	234 EA
E-circumflex		210 D2	137 89	146 92		202 CA
e-grave	138 8A	138 8A	138 8A	138 8A	138 8A	232 E8
E-grave		212 D4	146 92	145 91		200 C8
e-umlaut	137 89	137 89		137 89	137 89	235 EB
E-umlaut		211 D3		148 94		203 CB
i-acute	161 A1	161 A1	161 A1		161 A1	237 ED
I-acute		214 D6	139 8B			205 CD
i-circumflex	140 8C	140 8C		140 8C	140 8C	238 EE
I-circumflex		215 D7		168 A8		206 CE
i-dotless		213 D5				
i-grave	141 8D	141 8D	141 8D		141 8D	236 EC
I-grave		222 DE	152 98[24]			204 CC
i-umlaut	139 8B	139 8B		139 8B	139 8B	239 EF
I-umlaut		216 D8		149 95		207 CF
n-tilde	164 A4	164 A4	164 A4		164 A4	241 F1
N-tilde	165 A5	165 A5	165 A5		165 A5	209 D1
o-acute	162 A2	162 A2	162 A2	162 A2	162 A2	243 F3
O-acute		224 E0	159 9F			211 D3
o-circumflex	147 93	147 93	147 93	147 93	147 93	244 F4
O-circumflex		226 E2	140 8C	153 99		212 D4
o-grave	149 95	149 95	149 95		149 95	242 F2
O-grave		227 E3	169 A9			210 D2
o-slash		155 9B			155 9B	248 F8
O-slash		157 9D			157 9D	216 D8
o-tilde		228 E4	148 94			245 F5
O-tilde		229 E5	153 99			213 D5
o-umlaut	148 94	148 94			148 94	246 F6

[24]CP860 I-grave is also erroneously listed as 139/8B in the IBM DOS 3.30 manual.

Table II-5 IBM PC and PS/2 Code Pages (continued)

Code Page *Character*	**CP437** *Dec Hex*	**CP850** *Dec Hex*	**CP860** *Dec Hex*	**CP863** *Dec Hex*	**CP865** *Dec Hex*	**Latin-1** *Dec Hex*
O-umlaut	153 99	153 99			153 99	214 D6
u-acute	163 A3	163 A3	163 A3	163 A3[25]	163 A3[25]	250 FA
U-acute		233 E9	150 96			218 DA
u-circumflex	150 96	150 96		150 96	150 96	251 FB
U-circumflex		234 EA		158 9E		219 DB
u-grave	151 97	151 97	151 97	151 97[26]	151 97[26]	249 F9
U-grave		235 EB	157 9D	157 9D		217 D9
u-umlaut	129 81	129 81	129 81	129 81	129 81	252 FC
U-umlaut	154 9A	154 9A	154 9A	154 9A	154 9A	220 DC
y-acute		236 EC				253 FD
Y-acute		237 ED				221 DD
y-umlaut	152 98	152 98			152 98	255 FF
German ss	225 E1[27]	225 E1	225 E1[27]	225 E1[27]	225 E1[27]	223 DF
Greek alpha	224 E0		224 E0	224 E0	224 E0	
Greek beta	225 E1	225 E1[28]	225 E1	225 E1	225 E1	
Greek delta	235 EB		235 EB	235 EB	235 EB	
Greek epsilon	238 EE		238 EE	238 EE	238 EE	
Greek fi	237 ED		237 ED	237 ED	237 ED	
Greek Fi	232 E8		232 E8	232 E8	232 E8	
Greek Gamma	226 E2		226 E2	226 E2	226 E2	
Greek mu	230 E6	230 E6	230 E6	230 E6	230 E6	
Greek pi	227 E3		227 E3	227 E3	227 E3	
Greek sigma	229 E5		229 E5	229 E5	229 E5	
Greek Sigma	228 E4		228 E4	228 E4	228 E4	

[25]CP863 and 865 also erroneously list u-acute as 151/97.

[26]CP863 and 865 erroneously list u-acute in this position.

[27]Use Greek beta.

[28]Use German double s.

Table II-5 IBM PC and PS/2 Code Pages (continued)

Code Page *Character*	**CP437** *Dec Hex*	**CP850** *Dec Hex*	**CP860** *Dec Hex*	**CP863** *Dec Hex*	**CP865** *Dec Hex*	**Latin-1** *Dec Hex*
Greek tau	231 E7		231 E7	231 E7	231 E7	
Greek Theta	233 E9		233 E9	233 E9	233 E9	
Greek Omega	234 EA		234 EA	234 EA	234 EA	
Icelandic eth		208 D0				240 F0
Icelandic Eth		209 D1				208 D0
Icelandic thorn		231 E7				254 FE
Icelandic Thorn		232 E8				222 DE
English Pound	156 9C	156 9C	156 9C	156 9C	156 9C	163 A3
Japanese Yen	157 9D	190 BE				165 A5
Fem. ordinal	166 A6	166 A6	166 A6		166 A6	170 AA
Masc. ordinal	167 A7	167 A7	167 A7		167 A7	186 BA
!-inverted	173 AD	173 AD	173 AD		173 AD	161 A1
?-inverted	168 A8	168 A8	168 A8		168 A8	191 BF

MS-DOS Kermit Keyboard Scan Codes

Table II-6 shows the scan codes recognized by MS-DOS Kermit on the IBM PC and PS/2 USA keyboards, for use with Kermit's SET KEY command. The columns show the scan code for the key when pressed by itself, with Shift, with Ctrl, with Ctrl and Shift (C-S), with Alt, with Shift and Alt (S-A), with Ctrl and Alt (C-A), and with Ctrl, Shift, and Alt (C-S-A) all at the same time. In the *Key* column, kp refers to the numeric keypad, and Gray indicates the group of keys between the main keyboard and the numeric keypad on enhanced keyboards.

Keys marked with (*) used in conjunction with Ctrl and Ctrl-Shift will produce either the scan codes indicated or no scan code at all, depending on the keyboard model and driver. Older keyboards have no F11 and F12 keys.

Table II-6 Kermit Keyboard Scan Codes for the IBM PC and PS/2

Key	*Self*	*Shift*	*Ctrl*	*C-S*	*Alt*	*S-A*	*C-A*	*C-S-A*
' "	39	34			2344	2856	3368	3880
, <	44	60			2355	2867	3379	3891
- _	45	95	31	31	2434	2946	3458	3970
. >	46	62			2356	2868	3380	3892
/ ?	47	63			2357	2869	3381	3893
0) (*)	48	41	1464	1976	2433	2945	3457	3969
1 ! (*)	49	33	1465	1977	2424	2936	3448	3960
2 @	50	64	0	1795	2425	2937	3449	3961
3 # (*)	51	35	1467	1979	2426	2938	3450	3962
4 $ (*)	52	36	1468	1980	2427	2939	3451	3963
5 % (*)	53	37	1469	1981	2428	2940	3452	3964
6 ^	54	94	30	30	2429	2941	3453	3965
7 & (*)	55	38	1471	1983	2430	2942	3454	3966
8 * (*)	56	42	1472	1984	2431	2943	3455	3967
9 ((*)	57	40	1473	1985	2432	2944	3456	3968
; :	59	58			2343	2855	3367	3879
= +	61	43			2435	2947	3459	3971
[{	91	123	27	27	2330	2842	3354	3866

Table II-6 Kermit Keyboard Scan Codes for the IBM PC and PS/2 (continued)

Key	*Self*	*Shift*	*Ctrl*	*C-S*	*Alt*	*S-A*	*C-A*	*C-S-A*
\ |	92	124	28	28	2347	2859	3371	3883
] }	93	125	29	29	2331	2843	3355	3867
` ~	96	126			2345	2857	3369	3881
a A	97	65	1	1	2334	2846	3358	3870
b B	98	66	2	2	2352	2864	3376	3888
c C	99	67	3	3	2350	2862	3374	3886
d D	100	68	4	4	2336	2848	3360	3872
e E	101	69	5	5	2322	2834	3346	3858
f F	102	70	6	6	2337	2849	3361	3873
g G	103	71	7	7	2338	2850	3362	3874
h H	104	72	8	8	2339	2851	3363	3875
i I	105	73	9	9	2327	2839	3351	3863
j J	106	74	10	10	2340	2852	3364	3876
k K	107	75	11	11	2341	2853	3365	3877
l L	108	76	12	12	2342	2854	3366	3878
m M	109	77	13	13	2354	2866	3378	3890
n N	110	78	14	14	2353	2865	3377	3889
o O	111	79	15	15	2328	2840	3352	3864
p P	112	80	16	16	2329	2841	3353	3865
q Q	113	81	17	17	2320	2832	3344	3856
r R	114	82	18	18	2323	2835	3347	3859
s S	115	83	19	19	2335	2847	3359	3871
t T	116	84	20	20	2324	2836	3348	3860
u U	117	85	21	21	2326	2838	3350	3862
v V	118	86	22	22	2351	2863	3375	3887
w W	119	87	23	23	2321	2833	3345	3857
x X	120	88	24	24	2349	2861	3373	3885
y Y	121	89	25	25	2325	2837	3349	3861
z Z	122	90	26	26	2348	2860	3372	3884
Backspace	270	782	127	127	2318	2830	3342	3854

Table II-6 Kermit Keyboard Scan Codes for the IBM PC and PS/2 (continued)

Key	*Self*	*Shift*	*Ctrl*	*C-S*	*Alt*	*S-A*	*C-A*	*C-S-A*
Enter	284	796	10	10	2332	2844	3356	3868
Esc	27	27	27	27	2305	2817	3329	3841
F1	315	852	1374	1886	2408	2920	3432	3944
F2	316	853	1375	1887	2409	2921	3433	3945
F3	317	854	1376	1888	2410	2922	3434	3946
F4	318	855	1377	1889	2411	2923	3435	3947
F5	319	856	1378	1890	2412	2924	3436	3948
F6	320	857	1379	1891	2413	2925	3437	3949
F7	321	858	1380	1892	2414	2926	3438	3950
F8	322	859	1381	1893	2415	2927	3439	3951
F9	323	860	1382	1894	2416	2928	3440	3952
F10	324	861	1383	1895	2417	2929	3441	3953
F11	389	903	1417	1929	2443	2955	3467	3979
F12	390	904	1418	1930	2444	2956	3468	3980
Gray Delete	4435	4947	5523	6035	2467	2979		
Gray Down-Arrow	4432	4944	5521	6033	2464	2976	3488	4000
Gray End	4431	4943	5493	6005	2463	2975	3487	3999
Gray Home	4423	4935	5495	6007	2455	2967	3479	3991
Gray Insert	4434	4946	5522	6034	2466	2978	3490	4002
Gray Left-Arrow	4427	4939	5491	6003	2459	2971	3483	3995
Gray Page Down	4433	4945	5494	6006	2465	2977	3489	4001
Gray Page Up	4425	4937	5508	6020	2457	2969	3481	3993
Gray Right-Arrow	4429	4941	5492	6004	2461	2973	3485	3997
Gray Up-Arrow	4424	4936	5517	6029	2456	2968	3480	3992
Num Lock								
Pause/Break			1280	1792				
PrintScn/SysRq			1394	1906			3328	3840
Scroll Lock								
Space	32	32	32	32	32	32	32	32
Tab	271	783	1428	1940	2469	2981	3493	4005

Table II-6 Kermit Keyboard Scan Codes for the IBM PC and PS/2 (continued)

Key	*Self*	*Shift*	*Ctrl*	*C-S*	*Alt*	*S-A*	*C-A*	*C-S-A*
kp *	311	823	1430	1942	2359	2871	3383	3895
kp +	334	846	1424	1936	2382	2894	3406	3918
kp - (minus)	330	842	1422	1934	2378	2890	3402	3914
kp ., Del	339	851	1427	1939				
kp /	4399	4911	1429	1941	2468	2980	3492	4004
kp 0, Ins	338	850	1426	1938	**	**	**	**
kp 1, End	335	847	1397	1909	**	**	**	**
kp 2, Down-Arrow	336	848	1425	1937	**	**	**	**
kp 3, PgDn	337	849	1398	1910	**	**	**	**
kp 4, Left-Arrow	331	843	1395	1907	**	**	**	**
kp 5	332	844	1423	1935	**	**	**	**
kp 6, Right-Arrow	333	845	1396	1908	**	**	**	**
kp 7, Home	327	839	1399	1911	**	**	**	**
kp 8, Up-Arrow	328	840	1421	1933	**	**	**	**
kp 9, PgUp	329	841	1412	1924	**	**	**	**
kp Enter	4365	4877	5386	5898	2470	2982	3494	4006

The keypad digit keys 0–9 produce the codes listed only when Num Lock is off, except for Alt-key combinations which work the same whether Num Lock is on or off.

Alt-key combinations with keypad digits (marked ** above) produce scan codes based on the digits you type, but only after the Alt key is released. You may press one, two, or three keypad digit keys to enter any value from 1 to 255.

CAUTION: Pressing any of the combinations `Ctrl-Alt-Del`, `Ctrl-Alt-Delete`, `Ctrl-Alt-Shift-Del`, or `Ctrl-Alt-Shift-Delete` will reboot your PC.

Trademarks

Adobe Systems Incorporated: Postscript.
Apple Computer, Cupertino, CA: Apple II, Macintosh.
Ashton-Tate Corporation, Torrance, CA: dBase, dBase III, dBase III Plus, dBase IV.
AT&T Information Systems, Morristown, NJ: Bell, UNIX, Touch-Tone, Starlan.
Bibliographic Retrieval Services, Latham, NY: BRS.
Compaq Computer Corporation: COMPAQ.
CompuServe, Inc (an H&R Block Company), Columbus, OH: CompuServe.
Data General Corporation, Westboro, MA: DG/1.
Dialog Information Services, Inc.: Palo Alto, CA: DIALOG.
Digital Equipment Corporation, Maynard, MA: The Digital logo, DEC, PDP-11, VAX, VMS, VT52, VT100, VT102, VT320, Rainbow, VAXmate, DECserver, DECstation.
Dow Jones and Company, Princeton, NJ: Dow-Jones News/Retrieval Service.
Hayes Microcomputer Products, Inc., Norcross, GA: Hayes Smartmodem 1200 and 2400.
Heath Company, Benton Harbor, MI: Heath-19.
Henson Associates, Inc., New York, NY: Kermit. The Kermit protocol was named after Kermit the Frog, star of THE MUPPET SHOW television series. The name "Kermit" is used by permission of Henson Associates, Inc., New York.
Hercules Computer Technology: Hercules.
Hewlett-Packard Co., Palo Alto, CA: HP-150.
Intel Corp., Santa Clara, CA: Intel 8080, 8086, 8088, 80286, 80386, 80486, OpenNET.
International Business Machines Corp., Armonk, NY: IBM, Series/1, VM/CMS, MVS/TSO, PC-DOS, 3270, 3705, 3725, System/370, IBM PC, IBM PC/XT, IBM PC/AT, IBM PC*jr*, IBM RT PC, 7171, Netbios, Token Ring, TopView.
Lotus Development Corporation: Lotus 1-2-3.
MCI Communication Corporation: MCI Mail.
Microsoft Corporation, Bellevue, WA: MS-DOS, Xenix, MASM, Microsoft Windows.
Novell, Inc., Provo, UT: Novell Network.

Olivetti: Olivetti.
Oracle Corporation, Belmont, CA: Oracle.
Prime Computer, Natick, MA: PRIME, PRIMOS.
Sun Microsystems, Inc., Mountain View, CA: Sun Microsystems, Sun Workstation, SUN-4.
Telecom Canada, Ottawa, ON, Canada: Datapac.
Telenet Communications Corporation, Reston, VA: Telenet.
Teletype Corporation, Skokie, IL: Teletype.
TYMNET, Inc., San Jose, CA: TYMNET.
Ungermann-Bass, Santa Clara, CA: Net/One.
WordPerfect Corporation, Orem, UT: WordPerfect.
Xerox Corporation, Stamford, CT: Ethernet.
Zenith Data Systems, Glenview, IL: Zenith-19 Terminal.

The text of this book was entered using MS-DOS Kermit 3.0 on IBM PC/ATs and PS/2s into GNU EMACS on a SUN-4 UNIX system. The text was formatted for Postscript using the Scribe document preparation system. GNU EMACS is a product of the Free Software Foundation, 675 Massachusetts Avenue, Cambridge, MA 02139, USA. MS-DOS Kermit is a product of Kermit Distribution, Columbia University Center for Computing Activities, 612 West 115th Street, New York, NY 10025, USA. Scribe is a commercial product of Scribe Systems, Inc., Suite 240, Commerce Court, Four Station Square, Pittsburgh, PA 15219-1119, USA.

Index


BREAK AT SEAL AND PULL TO OPEN

"USING MS-DOS KERMIT covers all the topics needed to start new users easily, and it does so in a natural progression. . . . The pace is a lot faster than readers will realize, mostly because the writing is skillfully done—crisp, often entertaining, and always to the point. This is a terrific book!"

Joe R. Doupnik
Utah State University

This step-by-step guide shows you how to use MS-DOS Kermit Version 3.0 to connect your PC to the world. With the MS-DOS Kermit program diskette (included with this book), a cable, and possibly a modem, you can introduce your isolated PC to information and electronic services like Compuserve, Dow Jones News/Retrieval, or MCI Mail, to databases like BBS or DIALOG, to public data networks like Telenet and TYMNET. And most of all, to other computers: the PC on your desk to the corporate mainframe, your PC at home to the computers at work, your traveling laptop back to company headquarters, the PC in your dormitory room to the university computer center, the PC in your lab to a number-crunching supercomputer.

Included with this book is a Kermit Version 3.0 program diskette. It provides error-free file transfer, advanced terminal emulation, graphics, a script programming language, operation over local networks, support for text in many languages, special features for the disabled, and much more.

With the Kermit program and this book you now have a passport to the world of electronic communication and information!

digital™

Digital Press
Digital Equipment Corporation
12 Crosby Drive
Bedford, MA 01730

ORDER NUMBER EY-C204E-DP
DP ISBN 1-55558-048-3
PH ISBN 0-13-932476-3